

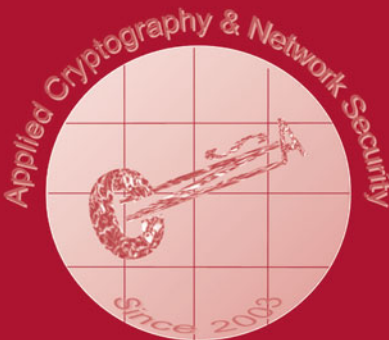
Mehdi Tibouchi
XiaoFeng Wang (Eds.)

LNCS 13906

Applied Cryptography and Network Security

21st International Conference, ACNS 2023
Kyoto, Japan, June 19–22, 2023
Proceedings, Part II

2
Part II



 Springer

MOREMEDIA



Lecture Notes in Computer Science

13906

Founding Editors

Gerhard Goos
Juris Hartmanis

Editorial Board Members

Elisa Bertino, *Purdue University, West Lafayette, IN, USA*

Wen Gao, *Peking University, Beijing, China*

Bernhard Steffen , *TU Dortmund University, Dortmund, Germany*

Moti Yung , *Columbia University, New York, NY, USA*

The series Lecture Notes in Computer Science (LNCS), including its subseries Lecture Notes in Artificial Intelligence (LNAI) and Lecture Notes in Bioinformatics (LNBI), has established itself as a medium for the publication of new developments in computer science and information technology research, teaching, and education.

LNCS enjoys close cooperation with the computer science R & D community, the series counts many renowned academics among its volume editors and paper authors, and collaborates with prestigious societies. Its mission is to serve this international community by providing an invaluable service, mainly focused on the publication of conference and workshop proceedings and postproceedings. LNCS commenced publication in 1973.

Mehdi Tibouchi · XiaoFeng Wang
Editors

Applied Cryptography and Network Security

21st International Conference, ACNS 2023
Kyoto, Japan, June 19–22, 2023
Proceedings, Part II

Editors

Mehdi Tibouchi 
NTT Social Informatics Laboratories
Tokyo, Japan

XiaoFeng Wang 
Indiana University at Bloomington
Bloomington, IN, USA

ISSN 0302-9743

ISSN 1611-3349 (electronic)

Lecture Notes in Computer Science

ISBN 978-3-031-33490-0

ISBN 978-3-031-33491-7 (eBook)

<https://doi.org/10.1007/978-3-031-33491-7>

© The Editor(s) (if applicable) and The Author(s), under exclusive license
to Springer Nature Switzerland AG 2023

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

ACNS 2023, the 21st International Conference on Applied Cryptography and Network Security, was held in Kyoto, Japan on June 19–22, 2023. The conference covered all technical aspects of applied cryptography, cyber security (including network and computer security) and privacy, representing both academic research works as well as developments in industrial and technical frontiers.

We received a total of 263 submissions from all over the world, among which the Program Committee (PC) selected 53 papers for publication in the proceedings of the conference. The two program chairs were supported by a PC consisting of 74 leading experts in all aspects of applied cryptography and security. Each submission received around 4 reviews from the committee. Strong conflict of interest rules ensured that papers were not handled by PC members with a close personal or professional relationship with the authors. The two program chairs were not allowed to submit a paper. There were approximately 180 external reviewers, whose input was critical to the selection of papers.

The review process was conducted using double-blind peer review. The conference had two possible submission deadlines, in September and January respectively. The authors of some submissions rejected from the September deadline, considered promising nonetheless, were encouraged to resubmit to the January deadline after appropriate revisions. Most of these revised papers were eventually accepted.

Alongside the presentations of the accepted papers, the program of ACNS 2023 featured two excellent invited talks by Shuichi Katsumata and Michalis Polychronakis.

The two volumes of the conference proceedings contain the revised versions of the 53 papers that were selected, together with the abstracts of the two invited talks. The final revised versions of papers were not reviewed again and the authors are responsible for their contents.

Following a long tradition, ACNS gives a *best student paper award* to encourage promising students to publish their best results at the conference. This year, the award was shared between two papers, one on the applied cryptography side and another on the security and systems side. The full-time students who received the awards were Agathe Cherièr for her paper “BIKE Key-Recovery: Combining Power Consumption Analysis and Information-Set Decoding” (co-authored with Nicolas Aragon, Tania Richmond and Benoît Gérard) and Ping-Lun Wang and Kai-Hsiang Chou for their paper “Capturing Antique Browsers in Modern Devices: A Security Analysis of Captive Portal Mini-Browsers” (co-authored with Shou-Ching Hsiao, Ann Tene Low, Tiffany Hyun-Jin Kim and Hsu-Chun Hsiao). The recipients shared a monetary prize of 1,500 EUR generously sponsored by Springer.

Many people contributed to the success of ACNS 2023. We would like to thank the authors for submitting their research results to the conference. We are very grateful to the PC members and external reviewers for contributing their knowledge and expertise, and for the tremendous amount of work involved in reviewing papers and contributing to the discussions. We are greatly indebted to Chunhua Su and Kazumasa Omote, the General

Chairs, for their efforts and overall organization. We thank the steering committee for their direction and valuable advice throughout the preparation of the conference. We also thank the team at Springer for handling the publication of these conference proceedings, as well as Siyuan Tang for helping with the preparation of the proceedings volumes.

June 2023

Mehdi Tibouchi
XiaoFeng Wang

Organization

General Co-chairs

Chunhua Su
Kazumasa Omote

University of Aizu, Japan
University of Tsukuba, Japan

Program Committee Co-chairs

Mehdi Tibouchi
XiaoFeng Wang

NTT Social Informatics Laboratories, Japan
Indiana University Bloomington, USA

Steering Committee

Moti Yung
Jianying Zhou

Google, USA
Singapore University of Technology and Design,
Singapore

Program Committee

Cristina Alcaraz
Giuseppe Ateniese
Xavier Bonnetain
Carlo Brunetta
Matteo Campanelli
Ignacio Cascudo
Sudipta Chattopadhyay

University of Málaga, Spain
George Mason University, USA
Inria, CNRS, Université de Lorraine, France
Simula UiB, Norway
Protocol Labs, Denmark
IMDEA Software Institute, Spain
Singapore University of Technology and Design,
Singapore

Kai Chen

Institute of Information Engineering, Chinese
Academy of Sciences, China

Liqun Chen
Chen-Mou Cheng
Jaeseung Choi
Sherman S. M. Chow
Michele Ciampi
Mauro Conti

University of Surrey, UK
BTQ AG, Japan
Sogang University, South Korea
The Chinese University of Hong Kong, China
University of Edinburgh, UK
Padova University, Italy

Sandro Coretti	IOHK, Switzerland
Roberto Di Pietro	Hamad Bin Khalifa University, Qatar
F. Betül Durak	Microsoft Research, USA
Nico Döttling	CISPA, Germany
Thomas Espitau	PQShield, UK
Antonio Faonio	Eurecom, France
Oriol Farràs	Universitat Rovira i Virgili, Spain
Prastudy Fauzi	Nanyang Technological University, Singapore
Tommaso Gagliardoni	Kudelski Security, Switzerland
Chaya Ganesh	IISc Bangalore, India
Debin Gao	Singapore Management University, Singapore
Paolo Gasti	New York Institute of Technology, USA
Hsu-Chun Hsiao	National Taiwan University, Taiwan
Hongxin Hu	University at Buffalo, USA
Xinyi Huang	Hong Kong University of Science and Technology, China
Akiko Inoue	NEC, Japan
Hai Jin	Huazhong University of Science and Technology, China
Stefan Katzenbeisser	University of Passau, Germany
Virginie Lallemand	CNRS, France
Kwok Yan Lam	Nanyang Technological University, Singapore
Peeter Laud	Cybernetica AS, Estonia
Changmin Lee	KIAS, South Korea
Mengyuan Li	MIT CSAIL, USA
Zhenkai Liang	National University of Singapore, Singapore
Hilder Vitor Lima Pereira	COSIC, KU Leuven, Belgium
Xiapu Luo	The Hong Kong Polytechnic University, China
Bernardo Magri	University of Manchester, UK
Mark Manulis	Universität der Bundeswehr München, Germany
Chloe Martindale	University of Bristol, UK
Daniel Masny	Meta, USA
Atsuko Miyaji	Osaka University, Japan
Cristina Onete	Université de Limoges, France
Michele Orrù	UC Berkeley, USA
Elena Pagnin	University of Lund, Sweden
Divya Ravi	Aarhus University, Denmark
Francisco Rodríguez-Henríquez	Technology Innovation Institute, UAE
Mélissa Rossi	ANSSI, France
Reihaneh Safavi-Naini	University of Calgary, Canada
Santanu Sarkar	IIT Madras, India
Mark Simkin	Ethereum Foundation, Denmark

Purui Su	Institute of Software, Chinese Academy of Sciences, China
Akira Takahashi	University of Edinburgh, UK
Katsuyuki Takashima	Waseda University, Japan
Qiang Tang	University of Sydney, Australia
Yiannis Tselekounis	Carnegie Mellon University, USA
Daniele Venturi	Sapienza University of Rome, Italy
Damien Vergnaud	Sorbonne Université, France
Alexandre Wallet	Inria, CNRS, Université de Rennes, France
Cong Wang	City University of Hong Kong, China
Wenhao Wang	Institute of Information Engineering, Chinese Academy of Sciences, China
Xueqiang Wang	University of Central Florida, USA
Takuya Watanabe	NTT Social Informatics Laboratories, Japan
Zhaoyan Xu	Palo Alto Networks, USA
Kazuki Yoneyama	Ibaraki University, Japan
Fan Zhang	Zhejiang University, China
Kehuan Zhang	The Chinese University of Hong Kong, China
Xiaokuan Zhang	George Mason University, USA
Hong-Sheng Zhou	Virginia Commonwealth University, USA
Jianying Zhou	Singapore University of Technology and Design, Singapore
Ruiyu Zhu	Meta, USA

Additional Reviewers

Aydin Abadi	Stefano Ceconello	Maryam Ehsanpour
Léo Ackermann	Debasmita Chakraborty	Nada El Kassem
Avishek Adhikari	Suvradip Chakraborty	Mojtaba Fadavi
Gora Adj	Jorge Chávez-Saab	Jiani Fan
Nabil Alkeilani Alkadri	Eyasu Getahun Chekole	Hanwen Feng
Ravi Anand	Jiageng Chen	Danilo Francati
Yan Lin Aung	Chenmou Cheng	Daniele Friolo
Sepideh Avizheh	Chengjun Lin	Ankit Gangwal
Christian Badertscher	Chihung Chi	Yingzi Gao
Zhenzhen Bao	Wonseok Choi	Daniel Gardham
Hugo Beguinet	Pratish Datta	Pierrick Gaudry
Vincenzo Botta	Luca Dolfi	Peter Gaži
Konstantinos Brazitikos	Denis Donadel	Robin Geelen
Maxime Buser	Minxin Du	Florian Gondesen
Andrea Caforio	Jesko Dujmovic	Antonio Guimarães
Matteo Cardaioli	Sabyasachi Dutta	Martin Gunnarsson

Jiale Guo	Kohei Nakagawa	Yosuke Todo
Keyan Guo	Thi Thu Quyen Nguyen	Junichi Tomida
Takuya Hayashi	Jianting Ning	Federico Turrin
Minki Hhan	Sabine Oechsner	Rei Ueno
Takato Hirano	Shinya Okumura	Serge Vaudenay
Akinori Hosoyamada	Luca Pajola	Jelle Vos
Elmo Huang	Ying-Yu Pan	Hendrik Waldner
Qiqing Huang	Giorgos Panagiotakos	Dawei Wang
Alberto Ibarrondo	Mahak Pancholi	Jiabo Wang
Yasuhiko Ikematsu	Bo Pang	Jiafan Wang
Takanori Isobe	Somnath Panja	Xiuhua Wang
Toshiyuki Isshiki	Jeongeun Park	Yalan Wang
Ryoma Ito	Lucas Perin	Yuntao Wang
Mahabir Prasad Jhanwar	Leo Perrin	Yohei Watanabe
Dingding Jia	Tran Phuong	Feng Wei
Xiangkun Jia	Sihang Pu	Harry W. H. Wong
Yanxue Jia	Rahul Rachuri	Baofeng Wu
Hao Jin	Mostafizar Rahman	Huangting Wu
Daniel Jost	Adrián Ranea	Keita Xagawa
Jiseung Kim	Prasanna Ravi	Yu Xia
Minkyu Kim	Chester Rebeiro	Jia Xu
Jakub Klemsa	Jordi Ribes González	Shengmin Xu
Dimitris Kolonelos	Leo Robert	Anshu Yadav
Yashvanth Kondi	Raghvendra Rohit	Takashi Yamakawa
Nishat Koti	Raghvendra Singh Rohit	Fan Yang
Jianchang Lai	Sergi Rovira	S. J. Yang
Meng Li	Dibyendu Roy	Zheng Yang
Yanan Li	Vishruta Rudresh	Takanori Yasuda
Yongzhi Lim	Rahul Saha	Weijing You
Chao Lin	Kosei Sakamoto	Thomas Zacharias
Chuanwei Lin	Matteo Salvino	Riccardo Zannotto
Fukang Liu	Pratik Sarkar	Bingsheng Zhang
Ziyao Liu	Michael Scott	Yangyong Zhang
Xiaojuan Lu	Siyu Shen	Zicheng Zhang
Jack P. K. Ma	Kazumasa Shinagawa	Yue Zhao
Damien Marion	Luisa Siniscalchi	Yafei Zheng
Christian Matt	Patrick Struck	Jiuqin Zhou
Kelsey Melissaris	Shiwei Sun	Zhelei Zhou
Guozhu Meng	Yao Sun	Vincent Zucca
Marine Minier	Younes Talibi Alaoui	Marius André Årdal
Alexander Munch-Hansen	Teik Guan Tan	Melek Önen
Mustafa A. Mustafa	Phuc Thai	Arne Tobias Ødegaard
Anne Müller	Yangguang Tian	
Misato Nakabayashi	Ivan Tjuawinata	

Contents – Part II

Embedded Security

A Forkcipher-Based Pseudo-Random Number Generator	3
<i>Elena Andreeva and Andreas Weninger</i>	
DMA'n'Play: Practical Remote Attestation Based on Direct Memory Access	32
<i>Sebastian Surminski, Christian Niesler, Lucas Davi, and Ahmad-Reza Sadeghi</i>	
Recommendation for a Holistic Secure Embedded ISA Extension	62
<i>Florian Stolz, Marc Fyrbiak, Pascal Sasdrich, and Tim Güneysu</i>	
QuantumCharge: Post-Quantum Cryptography for Electric Vehicle Charging	85
<i>Dustin Kern, Christoph Krauß, Timm Lauser, Nouri Alnahawi, Alexander Wiesmaier, and Ruben Niederhagen</i>	

Privacy-Preserving Protocols

Constant-Round Multiparty Private Function Evaluation with (Quasi-) Linear Complexities	115
<i>Yongfeng Xu, Hanyu Jia, Xiangxue Li, Qiang Li, Yue Bao, and Xintian Hou</i>	
Predicate Private Set Intersection with Linear Complexity	143
<i>Yaxi Yang, Jian Weng, Yufeng Yi, Changyu Dong, Leo Yu Zhang, and Jianying Zhou</i>	
A Framework for UC Secure Privacy Preserving Biometric Authentication Using Efficient Functional Encryption	167
<i>Johannes Ernst and Aikaterini Mitrokotsa</i>	
Private Information Retrieval with Result Verification for More Servers	197
<i>Pengzhen Ke and Liang Feng Zhang</i>	

Isogeny-Based Cryptography

Practical Robust DKG Protocols for CSIDH 219
Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen

Efficient Isogeny Proofs Using Generic Techniques 248
Kelong Cong, Yi-Fu Lai, and Shai Levin

Low Memory Attacks on Small Key CSIDH 276
*Jesús-Javier Chi-Domínguez, Andre Esser, Sabrina Kunzweiler,
 and Alexander May*

Encryption

On the Complete Non-malleability of the Fujisaki-Okamoto Transform 307
Daniele Friolo, Matteo Salvino, and Daniele Venturi

Optimal Security Notion for Decentralized Multi-Client Functional
 Encryption 336
Ky Nguyen, Duong Hieu Phan, and David Pointcheval

Anonymous (Hierarchical) Identity-Based Encryption from Broader
 Assumptions 366
Huangting Wu and Sherman S. M. Chow

Publicly Auditable Functional Encryption 396
Vlasis Koutsos and Dimitrios Papadopoulos

Advanced Primitives

Robustly Reusable Fuzzy Extractors in a Post-quantum World 429
Amit Deo and Charlie Grover

Subversion-Resilient Authenticated Encryption Without Random Oracles 460
*Pascal Bemmam, Sebastian Berndt, Denis Diemert,
 Thomas Eisenbarth, and Tibor Jager*

Scored Anonymous Credentials 484
Sherman S. M. Chow, Jack P. K. Ma, and Tsz Hon Yuen

GeT a CAKE: Generic Transformations from Key Encapsulation
 Mechanisms to Password Authenticated Key Exchanges 516
*Hugo Beguin, Céline Chevalier, David Pointcheval, Thomas Ricosset,
 and Mélissa Rossi*

Multiparty Computation

Explicit and Nearly Tight Lower Bound for 2-Party Perfectly Secure FSS 541
Keitaro Hiwatashi and Koji Nuida

Multi-Theorem Fiat-Shamir Transform from Correlation-Intractable Hash
 Functions 555
Michele Ciampi and Yu Xia

Game-Theoretically Secure Protocols for the Ordinal Random Assignment
 Problem 582
T.-H. Hubert Chan, Ting Wen, Hao Xie, and Quan Xue

A New Approach to Garbled Circuits 611
*Anasuya Acharya, Tomer Ashur, Efrat Cohen, Carmit Hazay,
 and Avishay Yanai*

Blockchain

Mt. Random: Multi-tiered Randomness Beacons 645
Ignacio Cascudo, Bernardo David, Omer Shlomovits, and Denis Varlakov

Revisiting Transaction Ledger Robustness in the Miner Extractable Value
 Era 675
Fredrik Kamphuis, Bernardo Magri, Ricky Lamberty, and Sebastian Faust

An Empirical Analysis of Security and Privacy Risks in Android
 Cryptocurrency Wallet Apps 699
I. Wayan Budi Sentana, Muhammad Ikram, and Mohamed Ali Kaafar

Author Index 727

Contents – Part I

Side-Channel and Fault Attacks

Formal Verification of Arithmetic Masking in Hardware and Software	3
<i>Barbara Gigerl, Robert Primas, and Stefan Mangard</i>	
Layered Binary Templating	33
<i>Martin Schwarzl, Erik Kraft, and Daniel Gruss</i>	
HS-Based Error Correction Algorithm for Noisy Binary GCD Side-Channel Sequences	59
<i>Kenta Tani and Noboru Kunihiro</i>	
Divide and Rule: DiFA - Division Property Based Fault Attacks on PRESENT and GIFT	89
<i>Anup Kumar Kundu, Shibam Ghosh, Dhiman Saha, and Mostafizar Rahman</i>	

Symmetric Cryptanalysis

A Novel Automatic Technique Based on MILP to Search for Impossible Differentials	119
<i>Yong Liu, Zejun Xiang, Siwei Chen, Shasha Zhang, and Xiangyong Zeng</i>	
Meet-in-the-Filter and Dynamic Counting with Applications to SPECK	149
<i>Alex Biryukov, Luan Cardoso dos Santos, Je Sen Teh, Aleksei Udovenko, and Vesselin Velichkov</i>	
Near Collision Attack Against Grain V1	178
<i>Subhadeep Banik, Daniel Collins, and Willi Meier</i>	
TIDAL: Practical Collisions on State-Reduced KECCAK Variants	208
<i>Sahiba Suryawanshi, Dhiman Saha, and Shashwat Jaiswal</i>	

Web Security

Tiny WFP: Lightweight and Effective Website Fingerprinting via Wavelet Multi-Resolution Analysis	237
<i>Cong Tian, Dengpan Ye, and Chuanxi Chen</i>	

Capturing Antique Browsers in Modern Devices: A Security Analysis of Captive Portal Mini-Browsers	260
<i>Ping-Lun Wang, Kai-Hsiang Chou, Shou-Ching Hsiao, Ann Tene Low, Tiffany Hyun-Jin Kim, and Hsu-Chun Hsiao</i>	
Those Aren't Your Memories, They're Somebody Else's: Seeding Misinformation in Chat Bot Memories	284
<i>Conor Atkins, Benjamin Zi Hao Zhao, Hassan Jameel Asghar, Ian Wood, and Mohamed Ali Kaafar</i>	
Social Honey-pot for Humans: Luring People Through Self-managed Instagram Pages	309
<i>Sara Bardi, Mauro Conti, Luca Pajola, and Pier Paolo Tricomi</i>	
Elliptic Curves and Pairings	
Pairings in Rank-1 Constraint Systems	339
<i>Youssef El Housni</i>	
Binary Kummer Line	363
<i>Sabyasachi Karati</i>	
Generalised Asynchronous Remote Key Generation for Pairing-Based Cryptosystems	394
<i>Nick Frymann, Daniel Gardham, Mark Manulis, and Hugo Nartz</i>	
Homomorphic Cryptography	
PIE: p -adic Encoding for High-Precision Arithmetic in Homomorphic Encryption	425
<i>Luke Harmon, Gaetan Delavignette, Arnab Roy, and David Silva</i>	
Analysis and Prevention of Averaging Attacks Against Obfuscation Protocols	451
<i>Kilian Becher, J. A. Gregor Lagodzinski, Javier Parra-Arnau, and Thorsten Strufe</i>	
FLSwitch: Towards Secure and Fast Model Aggregation for Federated Deep Learning with a Learning State-Aware Switch	476
<i>Yunlong Mao, Ziqin Dang, Yu Lin, Tianling Zhang, Yuan Zhang, Jingyu Hua, and Sheng Zhong</i>	

Machine Learning

Fast and Efficient Malware Detection with Joint Static and Dynamic Features Through Transfer Learning	503
<i>Mao V. Ngo, Tram Truong-Huu, Dima Rabadi, Jia Yi Loo, and Sin G. Teo</i>	
Efficient Network Representation for GNN-Based Intrusion Detection	532
<i>Hamdi Friji, Alexis Olivereau, and Mireille Sarkiss</i>	
EVADE: Efficient Moving Target Defense for Autonomous Network Topology Shuffling Using Deep Reinforcement Learning	555
<i>Qisheng Zhang, Jin-Hee Cho, Terrence J. Moore, Dan Dongseong Kim, Hyuk Lim, and Frederica Nelson</i>	
Steal from Collaboration: Spy Attack by a Dishonest Party in Vertical Federated Learning	583
<i>Hongbin Chen, Chaohao Fu, and Na Ruan</i>	

Lattices and Codes

Forward Security of Fiat-Shamir Lattice Signatures	607
<i>Yang Tao, Rui Zhang, and Yunfeng Ji</i>	
Shorter and Faster Identity-Based Signatures with Tight Security in the (Q)ROM from Lattices	634
<i>Éric Sageloli, Pierre Pébereau, Pierrick Méaux, and Céline Chevalier</i>	
A Gapless Post-quantum Hash Proof System in the Hamming Metric	664
<i>Bénédict Tran and Serge Vaudenay</i>	
Spherical Gaussian Leftover Hash Lemma via the Rényi Divergence	695
<i>Hiroki Okada, Kazuhide Fukushima, Shinsaku Kiyomoto, and Tsuyoshi Takagi</i>	
BIKE Key-Recovery: Combining Power Consumption Analysis and Information-Set Decoding	725
<i>Agathe Cheriére, Nicolas Aragon, Tania Richmond, and Benoît Gérard</i>	
Author Index	749

Embedded Security



A Forkcipher-Based Pseudo-Random Number Generator

Elena Andreeva and Andreas Wening^(✉)

TU Wien, Vienna, Austria

{elena.andreeva,andreas.weninger}@tuwien.ac.at

Abstract. Good randomness is needed for most cryptographic applications. In practice pseudo-random number generators (PRNGs) are employed. `CTR_DRBG` is a popular choice and among the recommended PRNGs by NIST. It is defined for use with primitives like AES or TDEA, which are not always suited for lightweight applications.

In this work we propose `FCRNG`, a new PRNG, similar to `CTR_DRBG`, that is optimized for the lightweight setting (e.g. the Internet of Things). Our `FCRNG` construction utilizes the expanding and tweakable forkcipher primitive instantiated with `ForkSkinny`, which was introduced by Andreeva et al. at ASIACRYPT 2019. `FCRNG` employs internally a forkcipher-based counter-style mode `FCTR`. We propose two `FCTR` variants: `FCTR-c` for optimized speed and `FCTR-T` for optimized security. We then show that `FCRNG` with `ForkSkinny` can be 33% faster than `CTR_DRBG` when instantiated with the AES blockcipher. `FCRNG` achieves also a better security bound in the robustness security game - first introduced by Dodis et al. at CCS'13 and now the standard security goal for PRNGs. Contrary to the CRYPTO 2020 security bound by Hoang and Shen established for `CTR_DRBG`, the security of our construction with `FCTR-T` does not degrade with the length of the random inputs, nor the amount of requested output pseudorandom bits. `FCRNG` passes all tests of the NIST test suite for pseudorandom number generators.

1 Introduction

Randomness in Cryptography. Sampling random values is a ubiquitous necessity in cryptography. It is used for a plethora of cryptographic purposes like generating keys, initialization vectors, nonces, challenges, shuffling, among others. On the other hand, the proofs of most cryptographic protocols simply assume that all parties are able to generate uniformly random values. It hence makes sense to separate the proof of the protocol in question from the way the underlying randomness is sampled. Since many cryptographic systems do not have access to truly random sources, it is therefore important to formally study the process of randomness sampling for which pseudo-random number generators (PRNG) are used. If the resulting pseudo-random numbers only possess weak randomness, many cryptographic protocols become insecure [15].

PRNGs have been recognized as an important tool in cryptography for a long time. As early as 1984, Santha and Vazirani [26] describe how to extract

indistinguishable-from-random bitstrings from bitstrings with low entropy. As a possible application they propose PRNGs. In 1986, Blum, Blum and Shub [9] present a pseudo-random number generator based on a computationally hard problem. In 2002, Desai, Hevia and Yin [13] studied PRNGs and gave a security framework. They analyze the ANSI X9.17 PRNG and the FIPS 183 PRNG.

In 2005, Barak and Halevi [7] proposed a formal security model for PRNG. It allows the adversary to corrupt the state. To the best of our knowledge, it is the first work that formalizes the notion that a PRNG should recover when given good randomness in the so-called refresh algorithm.

In 2013, Dodis et al. [14] extended the model of Barak and Halevi by not requiring the PRNG to fully recover in a single call to the refresh algorithm. Instead they formalize the property that a PRNG can slowly accumulate randomness over the course of many calls to the refresh algorithm. In 2019, Woodage and Shumow [30] investigated the security properties of the three PRNG constructions, that are recommended by NIST in their standard SP 800-90A. They put a particular focus on HASH-DRBG and HMAC-DRBG.

In 2020, Hoang and Shen [17] focused on the other NIST standardized PRNG construction, CTR_DRBG, which is the most used PRNG among the NIST recommendations¹. Hoang and Shen [17] showed that CTR_DRBG is also provably secure. CTR_DRBG is specified only based on the AES or Triple-DES (TDEA) blockciphers. Both of these ciphers can be rather costly for small devices (e.g. in the IoT), in particular in terms of area or speed. Furthermore, the reseeding algorithm of CTR_DRBG appears to be inefficient due to the internal algorithm, which Hoang and Shen [17] called CtE and is classified as a condenser. The required number of primitive calls by CtE is more than 3 times the block length of the input.

Lightweight Applications. While CTR_DRBG is instantiated with AES or TDEA according to the NIST standard SP 800-90A, its design is generic and can hence also be used with other blockciphers. Indeed, the security proof of Hoang and Shen [17] for CTR_DRBG does not rely on the concrete instantiation with AES or TDEA. Given the wide-spread use of CTR_DRBG, the goals in our work are to at least preserve the security soundness of CTR_DRBG and to further adapt it to a more efficient and lightweight-friendly PRNG. Lightweight in this context refers to designing an optimized cryptographic algorithm for resource-constrained devices that offers high-throughput processing, and/or compact implementation, and/or using a low amount of energy, among others. Such algorithmic optimizations enable practical applications with less powerful devices in the context of pervasive computing (e.g. RFID tags, microprocessors in small devices, IoT, etc.).

A natural first step towards obtaining a lightweight CTR_DRBG variant is to replace generically its NIST recommended underlying primitive (AES or TDEA) with a lightweight one. As of 2022, SKINNY [8] has been standardized as a

¹ Cohney et al. [12] noted that 67.8% of all certified implementations from NIST’s Cryptographic Module Validation Program (CMVP) in 2019 supported CTR_DRBG, making it the most popular design among these certifications.

lightweight tweakable blockcipher in the ISO (ISO/IEC 18033-7:2022) standard, which makes it a suitable replacement primitive. A tweakable blockcipher differs from (a regular) blockcipher in that, in addition to the key and the message, it also takes a public tweak value as an input. A tweakable blockcipher is a collection of independent blockciphers, that is indexed by the tweak. Tweakability allows for designing more secure provable modes of operation when compared to regular blockciphers. As a lightweight primitive, the biggest advantage of SKINNY is its low area cost, in particular compared to AES (e.g. for threshold implementations, SKINNY-128-128 only requires 3780 GE compared to the 8119 GE of AES-128 [8]). At the same time, SKINNY maintains a high level of security and has been extensively cryptanalysed [4, 5, 25, 28, 31, 32]. When compared to other lightweight blockciphers like LED [16], PRESENT [10], and PICCOLO [27], among others, SKINNY performs similarly or better in terms of area, throughput and security, and is a good choice for a well-rounded lightweight (satisfying multiple criteria) primitive [8].

Forkciphers. While CTR_DRBG can be generically upgraded with SKINNY as an underlying primitive, we investigate further to show that forkciphers, and particularly their SKINNY-based ForkSkinny instantiation, allows us to achieve better efficiency (than CTR_DRBG with SKINNY) and security results via our new PRNG FCRNG design. Forkciphers were first introduced in [2]. They can output a ciphertext of length twice the input block length. The two output blocks can be seen as the result of two calls to two independent tweakable blockciphers at a cheaper cost (than two independent calls). Forkciphers were shown suitable for achieving beyond-birthday (above $n/2$ bit security level with an n -bit input block) security and efficient in authenticated encryption for short messages [2] and CTR-mode style encryption [1].

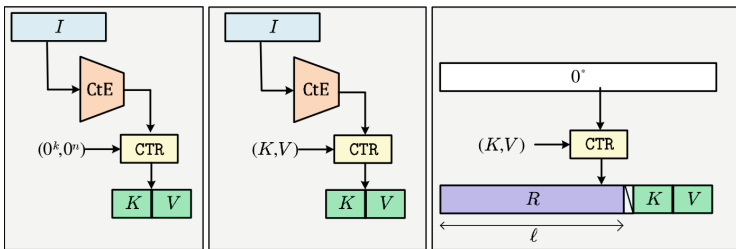


Fig. 1. The construction CTR_DRBG. The figure is a modified version of Fig. 5 in the work of Hoang and Shen [17]. The three diagrams show (from left to right) the algorithms $\text{setup}(I)$, $\text{refresh}(S, I)$, $\text{next}(S, \ell)$. In this construction $S = (K, V)$ represents the state. CTR denotes counter mode encryption. CtE is an algorithm that takes a random but non-uniform bitstring I of arbitrary size and returns an unpredictable bitstring of fixed length.

Contributions. In this work we present our construction FCRNG (ForkCipher Random Number Generator), which is based on the CTR_DRBG (Fig. 1), the most widely used PRNG from the NIST standard SP 800-90A. We make several improvements. First, we describe our PRNG generically based on a condenser. For the purpose we redefine the condenser security notion to explicitly support first block guessing. The condenser extracts an unpredictable fixed-size bitstring from an arbitrary length input bitstring that is random, but not uniform. We then propose a new condenser FCTRCond (depicted later in Fig. 4) which, as we show in Sect. 6, is approximately 60% faster than CtE, the condenser which was used in CTR_DRBG. Our efficiency improvement was achieved by employing a forkcipher as an underlying primitive. On the one hand, the forkcipher gives us more “packing” capability than a blockcipher since its tweak can also absorb additional input values. On the other hand, it requires less expansion primitive calls for the extracted value, since the forkcipher output length is twice as much as the output length of the blockcipher. In addition to the performance benefits, FCTRCond also comes with better security guarantees (i.e. the adversary has a lower chance to guess its outputs).

Next, the original CTR_DRBG is using CTR-mode encryption internally to compute the pseudorandom output bits and the next state of the PRNG. In our PRNG FCRNG we replace the CTR-mode with a forkcipher-based CTR variation called FCTR. We propose two FCTR instantiations, FCTR-c (later depicted in Fig. 5) and FCTR-T (later depicted in Fig. 6). For optimized speed we recommend the use of FCTR-c, whereas FCTR-T is better in terms of security, in particular when large pseudorandom data blocks are requested at once. FCTR-T bears similarities to GCTR-13 in [1] but does not use a nonce and instead uses a tweak that is the XOR result between the counter and a random value. Even though in their work Andreeva et al. concluded that GCTR-3 and GCTR-7 are the optimal variants for CTR with forkciphers, our FCTR-T construction comes closer to GCTR-13 due to the specificity of our PRNG setting where we do not have access to uniformly random values and because we also do not have easy access to nonces (especially since our security model allows the adversary to set the state i.e. they can force any number of nonce reuses). Still, we cannot apply the GCTR-13 security results generically, since the CTR variant is not used in a blackbox fashion in our proof. The efficiency improvement of FCTR compared to CTR-mode encryption with a SKINNY instantiation is 22% which is the result of replacing 2 tweakable blockcipher calls with 1 ForkSkinny call, equivalent to around 1.6 SKINNY calls altogether.

Our security analysis for the general construction FCRNG uses the analysis of CTR_DRBG by Hoang and Shen [17] as a reference due to their similar generic structure. As common, our security model considers the adversary and the distribution sampler, which is supposed to provide random inputs to our PRNG, as potentially malicious actors outside of the control of our construction. However, the distribution sampler must correctly specify the amount of entropy that is in the random bitstrings it produces. Like in the work of Hoang and Shen, the distribution sampler does not have access to the construction’s underlying prim-

itive, which is modeled as an ideal (fork)cipher. Our target security notion is robustness of PRNG, which was first introduced by Dodis et al. [14] and is now the standard security goal of PRNGs. It formalizes that even from a fully known state, the PRNG is able to go into an unpredictable state if enough entropy is supplied through reseeding. It implies backtracking and prediction resistance.

Our final FCRNG security bound drops several constant factors and entirely avoids two summands that are present in the security bound for CTR_DRBG [17], namely $\frac{48(\sqrt{q+1}) \cdot \sqrt{L+2} \cdot (\sigma+2p)}{2^n}$ (where L and σ are related to the length of the random inputs) and $\frac{2(B+3)(s+3p)}{2^n}$ (where B and s are related to the amount of pseudorandom output). This means the security of FCRNG with FCTR-T does not degrade with the length of the random inputs nor the amount of requested output pseudorandom bits.

In terms of overall efficiency, our construction FCRNG (instantiated with ForkSkinny and FCTR-c) is 28% faster than CTR_DRBG (with AES) for generating random bits, due to using FCTR instead of CTR-mode encryption and using ForkSkinny instead of AES. The usage of our condenser FCTRCond furthermore improves the efficiency of the initial setup and the reseeding algorithm refresh by approx. 72%. When an application reseeds once every 2000 bytes of requested pseudo-random output then we have an overall speedup of around 33%. This is discussed in detail in Sect. 6. FCRNG passes all tests of the NIST test suite for pseudorandom number generators, which can be found at [20].

2 Preliminaries

2.1 Notation

Let $a + b, a * b$ denote regular integer addition and multiplication. By $a \oplus b$ we denote bitwise XOR of two (equal length) bitstrings a, b . Let $\lceil r \rceil$ denote the smallest integer i s.t. $i \geq r$ for the real number r . $|a|$ denotes the length of bitstring a . By $[x]_y$ we denote encoding the value x as a bitstring of length y . $X[a : b]$ denotes the bitstring, that is obtained by taking bits $a, a + 1, \dots, b$ of bitstring X . $a||b$ denotes the concatenation of bitstrings a, b . Let $M_1, \dots, M_m \leftarrow^n M$ denote splitting a bitstring M , with $|M|$ being a multiple of n , into the blocks M_1, \dots, M_m ($\forall 1 \leq i \leq m : |M_i| = n$, hence $m = |M|/n$). $\text{Perm}(n)$ denotes the set of all permutations with range and domain $\{0, 1\}^n$.

Blockciphers and Forkciphers. A blockcipher E is defined as the pair of algorithms (E, E^{-1}) , where $E, E^{-1} : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ and it holds that $\forall K \in \{0, 1\}^k, M \in \{0, 1\}^n : E^{-1}(K, E(K, M)) = M$. We denote by k the key size and n the block size. By $\text{BC}(k, n)$ we denote the set of all such blockciphers.

Following [2], a forkcipher is a pair of deterministic algorithms, the forward encrypting and inverse algorithms, respectively:

$$F : \{0, 1\}^k \times \{0, 1\}^t \times \{0, 1\}^n \times \{0, 1, \mathbf{b}\} \rightarrow \{0, 1\}^n \cup \{0, 1\}^n \times \{0, 1\}^n$$

$$F^{-1} : \{0, 1\}^k \times \{0, 1\}^t \times \{0, 1\}^n \times \{0, 1\} \times \{\mathbf{i}, \mathbf{o}, \mathbf{b}\} \rightarrow \{0, 1\}^n \cup \{0, 1\}^n \times \{0, 1\}^n$$

Note that we use \cup in the definitions since the output can be a single n -bit string or a pair of such strings, depending on the chosen mode, as we define below. A forkcipher uses an additional tweak of size t . By $\text{FC}(k, t, n)$ we denote the set of all such forkciphers. A tweakable forkcipher F meets the *correctness condition*, if for every $K \in \{0, 1\}^k$, $\mathsf{T} \in \{0, 1\}^t$, $M \in \{0, 1\}^n$ and $\beta \in \{0, 1\}$ all of the following conditions are met:

1. $F^{-1}(K, \mathsf{T}, F(K, \mathsf{T}, M, \beta), \beta, \mathbf{i}) = M$
2. $F^{-1}(K, \mathsf{T}, F(K, \mathsf{T}, M, \beta), \beta, \mathbf{o}) = F(K, \mathsf{T}, M, \beta \oplus 1)$
3. $(F(K, \mathsf{T}, M, \mathbf{o}), F(K, \mathsf{T}, M, \mathbf{1})) = F(K, \mathsf{T}, M, \mathbf{b})$
4. $(F^{-1}(K, \mathsf{T}, C, \beta, \mathbf{i}), F^{-1}(K, \mathsf{T}, C, \beta, \mathbf{o})) = F^{-1}(K, \mathsf{T}, C, \beta, \mathbf{b})$

For each pair of key and tweak, the forkcipher applies two independent permutations to the input to produce the two output blocks. We use the shorthand $F_K^{T,s}(m) := F(K, T, m, s)$. Since most of our algorithms only use $s = \mathbf{b}$ we also use $F_K^T(m) := F_K^{T,\mathbf{b}}(m)$. Furthermore denote $(F^{-1})_K^{T,\beta,s}(c) := F^{-1}(K, T, c, \beta, s)$.

Almost Universal (AU) Hash. Let $H : \text{Seed} \times \text{Dom} \rightarrow \{0, 1\}^n$ be a (keyed) hash function. For each string X , define its *block length* to be $\max\{1, |X|/n\}$. For a function $\delta : \mathbb{N} \rightarrow [1, \infty)$, we say that H is a δ -almost universal hash if for every distinct strings X_1, X_2 whose block lengths are at most l , we have

$$\Pr_{\text{seed} \leftarrow \text{sSeed}} [H(\text{seed}, X_1) = H(\text{seed}, X_2)] \leq \frac{\delta(l)}{2^n}$$

Conditional Min-Entropy and Statistical Distance. For two random variables X and Y , the (*average-case*) *conditional min-entropy* of X given Y is

$$H_\infty(X|Y) = -\log\left(\sum_y \Pr[Y = y] * \max_x \Pr[X = x|Y = y]\right)$$

The *statistical distance* between two random variables X and Y is defined as

$$\text{SD}(X, Y) = \frac{1}{2} \sum_z |\Pr[X = z] - \Pr[Y = z]|$$

$\text{SD}(X, Y)$ is the best possible advantage of an (even computationally unbounded) adversary in distinguishing X and Y .

2.2 Systems, Transcripts and the H-coefficient Proof Technique

Following [17, 18] we consider the interactions of a distinguisher \mathcal{A} with an abstract system S which answers \mathcal{A} 's queries. The resulting interaction then generates a transcript $\tau = ((X_1, Y_1), \dots, (X_q, Y_q))$ of query-answer pairs. S is entirely described by the probabilities $\text{ps}(\tau)$ that correspond to the system S

responding with answers as indicated by τ when queries in τ are made. We will generally describe systems informally, or more formally in terms of a set of oracles they provide, and only use the fact that they define corresponding probabilities $\text{ps}(\tau)$ without explicitly giving these probabilities. We say that a transcript is valid for system S if $\text{ps}(\tau) > 0$.

For any systems S_1 and S_0 , let $\Delta_{\mathcal{A}}(S_1, S_0)$ denote the distinguishing advantage of the adversary \mathcal{A} against the “real” system S_1 and the “ideal” system S_0 .

Following [17], we now describe the H-coefficient technique of Patarin [11, 21]. Generically, it considers a deterministic distinguisher \mathcal{A} that tries to distinguish a “real” system S_1 from an “ideal” system S_0 . The adversary’s interactions with those systems define transcripts X_1 and X_0 , respectively, and a bound on the distinguishing advantage of \mathcal{A} is given by the statistical distance $\text{SD}(X_1, X_0)$.

Lemma 1 (see [11, 21]). *Suppose we can partition the set of valid transcripts for the ideal system into good and bad ones. Further, suppose that there exists $\epsilon \geq 0$ such that $1 - \frac{\text{ps}_1(\tau)}{\text{ps}_0(\tau)} \leq \epsilon$ for every good transcript τ . Then,*

$$\text{SD}(X_1, X_0) \leq \epsilon + \Pr[X_0 \text{ is bad}]$$

3 Security Model

In this section we describe the syntax and security notions related to our construction its building blocks. We introduce the ideal forkcipher model, analogously to the ideal cipher model and recall the notions of a condenser and a pseudorandom number generator.

Ideal (Fork)Cipher. In the ideal cipher model, a block cipher E is chosen from $\text{BC}(k, n)$ uniformly at random. It allows for two types of oracle queries $E(K, m)$ and $E^{-1}(K, c)$ for $m, c \in \{0, 1\}^n$ and $K \in \{0, 1\}^k$. The response to an inverse query $E^{-1}(K, c)$ is $m \in \{0, 1\}^n$ s.t. $E(K, m) = c$.

In the ideal forkcipher model, a forkcipher F is chosen from $\text{FC}(k, t, n)$ uniformly at random. The ideal primitive also provides oracle access to F and F^{-1} , allowing the adversary to query $F(K, T, m, s)$ and $F^{-1}(K, T, c, \beta, s')$ for $m, c \in \{0, 1\}^n, T \in \{0, 1\}^t, s \in \{0, 1, \mathbf{b}\}, s' \in \{\mathbf{i}, \mathbf{o}, \mathbf{b}\}, \beta \in \{0, 1\}$ and $K \in \{0, 1\}^k$. The adversary queries the ideal primitive with full control over all parameters.

PRNG. A Pseudorandom Number Generator (PRNG) with input I with state space State and seed space Seed is a tuple of deterministic algorithms $G = (\text{setup}, \text{refresh}, \text{next})$.

- $\text{setup}(\text{seed}, I)$ takes a seed $\text{seed} \in \text{Seed}$ and a string I as input, to then output an initial state $S \in \text{State}$.
- $\text{refresh}(\text{seed}, S, I)$ takes as input $\text{seed} \in \text{Seed}, S \in \text{State}$ and string I and then outputs a new state.

- $\text{next}(\text{seed}, S, l)$ takes as input $\text{seed} \in \text{Seed}$, $S \in \text{State}$ and a number $l \in \mathbb{N}$ to then output a new state and an l -bit output string.

Informally, the input I gives the necessary randomness to the otherwise deterministic algorithms. For our construction, the seed can be seen as a proof artifact that is not actually part of the construction. Indeed, as we discuss later, we will treat the seed as the full description of the ideal ciphers E and F .

Condensers. A condenser Cond [24] takes a bitstring that has low entropy (i.e. is not uniformly random) and outputs a value that is hard to predict. We follow [17] for the subsequent definitions. Let S be a λ -source, meaning a stateless, probabilistic algorithm that outputs a random input I and some side information z , such that $H_\infty(I|z) \geq \lambda$. For any adversary \mathcal{A} , we define the guessing advantage of \mathcal{A} against condenser Cond with a source S as

$$\text{Adv}_{\text{Cond}}^{\text{guess}}(\mathcal{A}, S) = \Pr[\mathbf{G}_{\text{Cond}}^{\text{guess}}(\mathcal{A}, S)]$$

The corresponding game is described in Fig. 2.

Game $\mathbf{G}_{\text{Cond}}^{\text{guess}}(\mathcal{A}, S)$	
	$(I, z) \leftarrow \$ S$; $\text{seed} \leftarrow \$ \text{Seed}$; $V \leftarrow \text{Cond}(\text{seed}, I)$
	$(Y_1, \dots, Y_q) \leftarrow \$ \mathcal{A}(\text{seed}, z)$; return $(V \in \{Y_1, \dots, Y_q\})$
Game $\mathbf{G}_{\text{Cond}}^{1\text{-blk-guess}}(\mathcal{A}, S)$	
	$(I, z) \leftarrow \$ S$; $\text{seed} \leftarrow \$ \text{Seed}$; $V \leftarrow \text{Cond}(\text{seed}, I)[1 : n]$
	$(Y_1, \dots, Y_q) \leftarrow \$ \mathcal{A}(\text{seed}, z)$; return $(V \in \{Y_1, \dots, Y_q\})$

Fig. 2. Security games for a condenser Cond

We introduce a modified notion about guessing the first block. The motivation for creating this new notion is that for the later security proofs, we require the first output block of Cond to be hard to guess. One could then show that for a Cond , when the output is truncated to n bits, it constitutes a secure condenser with respect to the $\mathbf{G}_{\text{Cond}}^{\text{guess}}(\mathcal{A}, S)$ notion. This was in fact implicitly used by Hoang and Shen in [17]. However, here we prove a property for a different component used in the overall construction. This becomes an issue when defining a generic construction that uses a condenser as building block, as is the case in this work. In this scenario, one wants a concise way to describe what is required of the condenser. Hence, for any adversary \mathcal{A} , we define the 1-block-guessing advantage of \mathcal{A} against condenser Cond with a source S as

$$\text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(\mathcal{A}, S) = \Pr[\mathbf{G}_{\text{Cond}}^{1\text{-blk-guess}}(\mathcal{A}, S)]$$

The corresponding security game is also described Fig. 2.

Distribution Samplers. Distribution samplers are similar to λ -sources, but are stateful and give an estimate of how much entropy they provide. A distribution sampler \mathcal{D} is a stateful, probabilistic algorithm. Given the current state s , it will output a tuple (s', I, γ, z) in which s' is the updated state and I is the next randomness input for the PRNG G . $\gamma \geq 0$ is a real number that, informally speaking, will tell us the amount of entropy in I . z is some side information of I given to an adversary attacking G . Let p be an upper bound of the number of calls to \mathcal{D} in our security games. Let s_0 be the empty string, and let $(s_i, I_i, \gamma_i, z_i) \leftarrow {}_s \mathcal{D}(s_{i-1})$ for every $i \in \{1, \dots, p\}$. For each $i \leq p$, let

$$\mathcal{I}_{p,i} = (I_1, \dots, I_{i-1}, I_{i+1}, \dots, I_p, \gamma_1, \dots, \gamma_p, z_1, \dots, z_p)$$

We say that sampler \mathcal{D} is *legitimate* if $H_\infty(I_i | \mathcal{I}_{p,i}) \geq \gamma_i$ for every $i \in \{1, \dots, p\}$. A legitimate sampler is λ -simple if $\gamma_i \geq \lambda$ for every i . Following [17], in this work we will only consider simple samplers for a sufficiently large min-entropy threshold λ . As they noted, this is somewhat limiting as it fails to show that the PRNG can slowly accumulate randomness by absorbing many low entropy inputs. However, the results are still meaningful and is the setting that was considered in the NIST SP 800-90A standard.

Robustness. The game $\mathbf{G}_{G,\lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D})$ is defined in Fig. 3. It is played for a PRNG $G = (\text{setup}, \text{refresh}, \text{next})$, an adversary \mathcal{A} and a distribution sampler \mathcal{D} , with respect to an entropy threshold λ . The internal variable c counts the current amount of entropy. While it is too low, the oracle RoR is designed to be useless to the adversary. Define

$$\text{Adv}_{G,\lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D}) = 2 * \Pr[\mathbf{G}_{G,\lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D})] - 1$$

4 Our Constructions

In this section we describe our condenser FCTRCond and our PRNG FCRNG.

4.1 A Forkcipher-Based Condenser

Our Construction. FCTRCond is described with respect to a constant v , which must be chosen s.t. $0 \leq v \leq t - 2$. Let $\text{pad}^* : \{0, 1\}^* \rightarrow (\{0, 1\}^{n+k+v})^+$ be the padding scheme that appends the byte $0x59$ and then appends 0's until the length is a multiple of $k + n + v$. Note that $\text{pad}^*(X) \neq \text{pad}^*(Y)$ for any $X \neq Y$. We describe FCTRCond based on a forkcipher F in Fig. 4.

Tweak Packing Constant v . The constant v determines how many bits of the tweak of the internal forkcipher are used to absorb data. Large values of v increase the efficiency of FCTRCond, but decrease the amount of input data that can be processed at once, as it reduces the available bits to encode the block index. Specifically, FCTRCond can process inputs I with maximum length up to $2^{t-v-1} \cdot (k + n + v)$. As a general purpose value one may set $v = \lceil t/2 \rceil$. For our efficiency estimation we assume that this is the case.

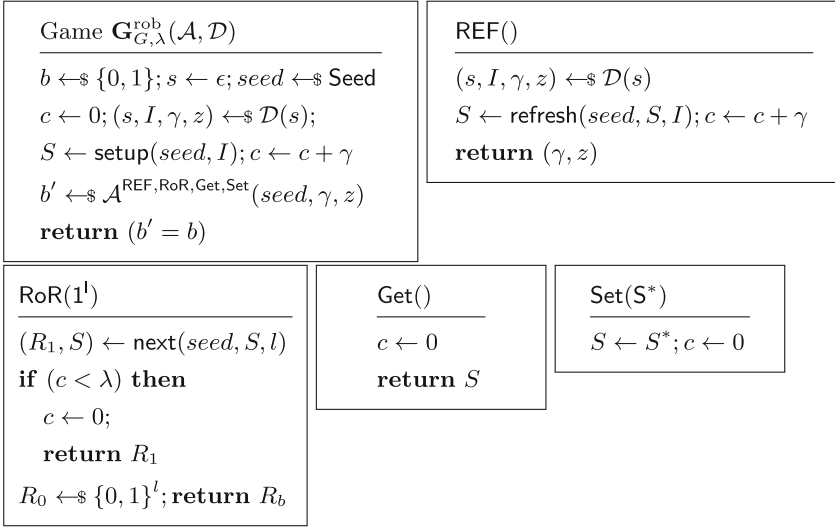


Fig. 3. Robustness game (see Fig. 1 in [17])

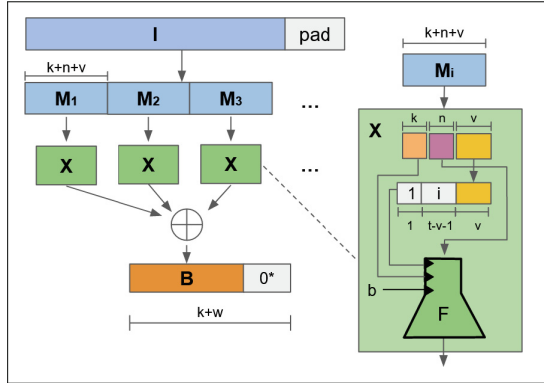
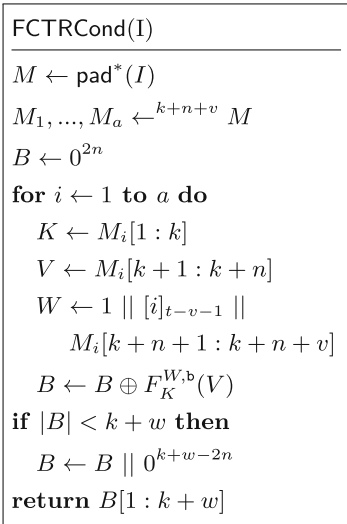


Fig. 4. The forkcipher-based condenser FCTRCond.

4.2 Our FCRNG PRNG Construction

In the following section we formally introduce the protocol $\text{FCRNG} = (\text{setup}, \text{refresh}, \text{next})$ which conforms to the notion of a robust PRNG. It is based on a secure condenser Cond which takes low entropy random inputs I and outputs unpredictable values. We will prove the security of FCRNG generically, and later provide concrete instantiation with FCTRCond for our efficiency evaluation.

- **setup** takes some random input I and applies Cond to it in order to initialize the state.
- **refresh** takes some random input I and applies Cond to it in order to update the state. Even if the state was fully known before, calling **refresh** (with sufficient entropy) makes the state unpredictable for the adversary.
- **next** outputs a pseudo-random value based on the current state. Afterwards the state is updated.

In all functions (**setup**, **refresh**, **next**) we use our algorithm FCTR to set the next state and also produce the pseudo-random output in **next**. It effectively creates an XOR-mask based on the current state and applies it to the output of Cond (or 0 in the case of **next**). FCTR was designed in 2 variations from which one can be chosen. Figure 5 shows FCTR-c , which is the variation with a higher throughput. Figure 6 shows FCTR-T , which is more secure.

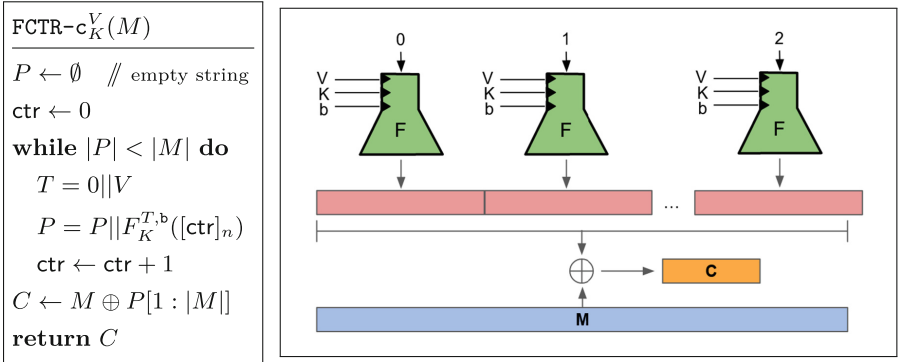


Fig. 5. Algorithm FCTR-c (Forkcipher CounTeR Classic). F denotes a forkcipher. The tweak starts with the 0 bit to ensure that it never occurs in FCTRCond .

Our construction FCRNG is shown in Fig. 7. It uses a forkcipher $F : \{0, 1\}^k \times \{0, 1\}^t \times \{0, 1\}^n \times \{0, 1, b\} \rightarrow \{0, 1\}^n \cup \{0, 1\}^n \times \{0, 1\}^n$. In $\text{FCRNG} = (\text{setup}, \text{refresh}, \text{next})$, the state consists of the cipher key $K \in \{0, 1\}^k$ and the value $V \in \{0, 1\}^w$ (w is defined below), which is used as an IV (initialization vector) for counter mode encryption. The construction does not rely on a seed, but instead on the forkcipher F . In the later proofs we will view F as an idealized primitive. As is common in the literature [17, 30], the seed will be treated as the full description of the ideal forkcipher F .

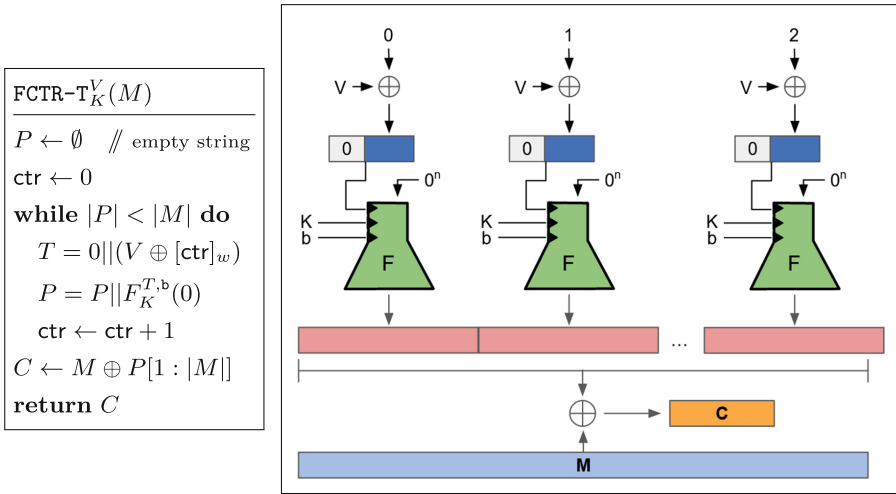


Fig. 6. Algorithm FCTR-T (Forkcipher CounTeR in Tweak). The tweak is set to the 0 bit string to ensure that it never reoccurs in FCTRCond.

Let $\text{pad} : \{0, 1\}^* \rightarrow (\{0, 1\}^n)^+$ be the padding scheme that appends the byte $0\mathbf{x}08$ and then appends 0's until the length is a multiple of n . Note that $\text{pad}(X) \neq \text{pad}(Y)$ for any $X \neq Y$.

The Constant w : We use $w = \min\{n, t\} - 1$. It is required that $w + 1 \leq t$ in order to use V together with the prefix bit 0 as tweak, as is done in FCTR-T. We restrict $w + 1 \leq n$ since larger values are not supported by our security bound (see Sect. 5) but would also result in a worse efficiency (see Sect. 6).

Note that w limits the maximum block length of requested pseudo-random bits. The reason is that next (specifically FCTR-T) XORs a counter, that increases every second block, into the tweak. Hence we will assume that when calling $\text{next}(S, l)$ it holds that $l \leq 2^{w+1} \cdot n - k - w$.

5 Security Proofs

In this section we will give the security proofs of the condenser FCTRCond and the PRNG FCRNG, which generically uses a condenser and can hence use FCTRCond. For the condenser CtE (used in CTR_DRBG) as well as its security proof, refer to the analysis of Hoang and Shen [17] and our discussion in the full version of this paper.

5.1 Security of FCTRCond

In this section we will prove the following theorem that formalizes the security of FCTRCond.

$\text{setup}^F(I)$	$\text{refresh}^F(S, I)$	$\text{next}^F(S, l)$
$X \leftarrow \text{Cond}(I)$	$X \leftarrow \text{Cond}(I)$	$K \leftarrow S[1 : k]; V \leftarrow S[k + 1 : k + w]$
$K \leftarrow 0^k; V \leftarrow 0^w$	$K \leftarrow S[1 : k]$	$r \leftarrow n * \lceil l/n \rceil$
$S \leftarrow \text{FCTR}_K^V(X)$	$V \leftarrow S[k + 1 : k + w]$	$P \leftarrow \text{FCTR}_K^V(0^{r+k+w})$
return S	$S \leftarrow \text{FCTR}_K^V(X)$	$R \leftarrow P[1 : l]$
	return S	$S \leftarrow P[r + 1 : r + k + w]$
		return (R, S)

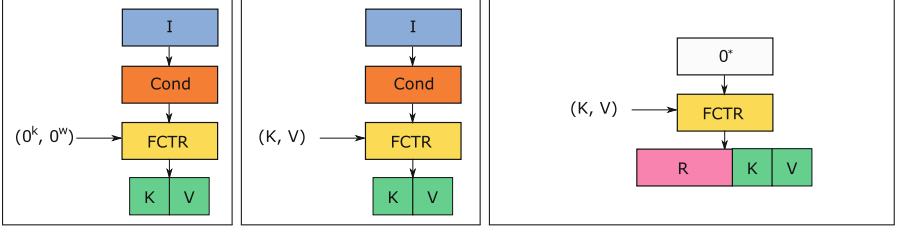


Fig. 7. The construction FCRNG. Cond denotes a condenser, F denotes a forkcipher. FCTR should be instantiated with FCTR-c (Fig. 5) or FCTR-T (Fig. 6). Depending on this instantiation we also refer to the construction as FCRNG-c or FCRNG-T, respectively.

Theorem 1. *Let F be a forkcipher that we model as an ideal forkcipher. Let FCTRCond be as described above. Let S be a λ -source that does not have access to F . Then for any adversary \mathcal{A} against FCTRCond in the 1-block-guessing game making at most q guesses has advantage at most*

$$\text{Adv}_{\text{FCTRCond}}^{1\text{-blk-guess}}(\mathcal{A}, S) \leq \frac{3q}{2^n} + \frac{\sqrt{q}}{2^{\lambda/2}}$$

We start by proving some intermediate lemmas. First we give a bound on the probability of the first block coinciding, when two different inputs are fed to FCTRCond. For more convenient notation in the following proofs, let FCTRCond^* denote the algorithm that runs FCTRCond but only returns the first block, i.e. $\text{FCTRCond}^*(I) := \text{FCTRCond}(I)[1 : n]$.

Lemma 2. *Let F be a forkcipher that we model as an an ideal forkcipher. Let FCTRCond* be as described above. Let I_1, I_2 be arbitrary strings s.t. $I_1 \neq I_2$. Then $\Pr[\text{FCTRCond}^*(I_1) = \text{FCTRCond}^*(I_2)] \leq \frac{1}{2^{n-1}}$ where the randomness is taken over the choices of F .*

Proof. Let $M = \text{pad}^*(I_1)$, $M' = \text{pad}^*(I_2)$. According to the second line of FCTRCond, let M_1, \dots, M_a and M'_1, \dots, M'_b be the result of splitting M and M' , respectively. Let B, B' be the output of $\text{FCTRCond}^*(I_1)$ and $\text{FCTRCond}^*(I_2)$, respectively.

Case 1: $a \neq b$. Without loss of generality, assume $a > b$. Let $X := F_K^{W,b}(V)[1 : n]$ with $K \leftarrow M_a[1 : k], V \leftarrow M_a[k + 1 : k + n], W \leftarrow 1 \parallel [a]_{t-v-1} \parallel M_a[k + n + 1 : k + n + v]$ (i.e. X is the a -th XOR-summand of B). Since no other call to F by

FCTRCond^{*}(I_1) or FCTRCond^{*}(I_2) uses this tweak, X is a uniformly random and independent variable. As a result, B , which XOR-sums over X , is also uniformly random and independent of B' . Hence $\Pr[B = B'] \leq \frac{1}{2^n} < \frac{1}{2^n - 1}$.

Case 2: $a = b$. Here $\Pr[B = B'] \leq \frac{1}{2^n - 1}$. This is easy to see, since this is a special case of Lemma 6 (see Appendix A) where $a' = a$ and $d = 0^n$. \square

As was done in previous works, we treat the full description of the ideal F as the equivalent to a seed. Therefore, Lemma 2 means that FCTRCond^{*} can be viewed as an δ -almost universal hash function. Since $\frac{1}{2^n - 1} = \frac{2^n}{2^n - 1}$, $\delta(l) = \frac{2^n}{2^n - 1}$ for all $l \in \mathbb{N}$.

Lemma 3 (Generalized Leftover Hash Lemma). [17] *Let Cond : Seed \times Dom $\rightarrow \{0, 1\}^n$ be a δ -AU hash function, and let $\lambda > 0$ be a real number. Let S be a λ -source whose random input I has at most l blocks. For any adversary \mathcal{A} making at most q guesses,*

$$\text{Adv}_{\text{Cond}}^{\text{guess}}(\mathcal{A}, S) \leq \frac{q}{2^n} + \sqrt{\frac{q}{2^\lambda} + \frac{q \cdot (\delta(l) - 1)}{2^n}}$$

The above formulation of Lemma 3 stems from Hoang and Shen [17] and was originally proven by Barak et al. [6]. Since we use the same security game $\mathbf{G}_{\text{Cond}}^{\text{guess}}(\mathcal{A}, S)$ as [17] we can apply this notation.

Lemma 4. *Let F be a forkcipher that we model as an ideal forkcipher. Let FCTRCond^{*} be as described above. Let S be a λ -source that does not have access to F . Then for any adversary \mathcal{A} against FCTRCond^{*} in the guessing game making at most q guesses has advantage at most*

$$\text{Adv}_{\text{FCTRCond}^*}^{\text{guess}}(\mathcal{A}, S) \leq \frac{3q}{2^n} + \frac{\sqrt{q}}{2^{\lambda/2}}$$

Proof. We use Lemma 3 and apply the result of Lemma 2.

$$\begin{aligned} \text{Adv}_{\text{FCTRCond}^*}^{\text{guess}}(\mathcal{A}, S) &\leq \frac{q}{2^n} + \frac{\sqrt{q}}{2^{\lambda/2}} + \sqrt{\frac{q \cdot (\frac{2^n}{2^n - 1} - 1)}{2^n}} \\ &\leq \frac{q}{2^n} + \frac{\sqrt{q}}{2^{\lambda/2}} + \frac{\sqrt{q}}{2^n - 1} \leq \frac{2q}{2^n - 1} + \frac{\sqrt{q}}{2^{\lambda/2}} \leq \frac{3q}{2^n} + \frac{\sqrt{q}}{2^{\lambda/2}} \quad (\text{since } n > 1) \end{aligned}$$

\square

The above lemma about FCTRCond^{*} immediately implies Theorem 1.

Discussion. When comparing this security bound to that of CtE (the condenser in the original CTR_DRBG, for the details refer to the the analysis by Hoang and Shen [17] or our discussion in the full version of this paper), we can see that FCTRCond is more secure, as the last summand of the security bound of CtE, $\frac{8\sqrt{q(l+2)^3}}{2^n}$, was avoided. This means that the security of FCTRCond does not degrade through the blocklength of the random inputs.

5.2 Security of FCRNG

We will prove that FCRNG-c and FCRNG-T are robust PRNGs. We consider an adversary \mathcal{A} that makes at most q oracle queries (including ideal-cipher queries). To be specific, the oracles are REF, RoR, Get, Set (see Fig. 3) and furthermore the adversary can make arbitrary calls to the ideal forkcipher F and any other ideals that are used by Cond (e.g. the ideal blockcipher E in the case of CtE from CTR_DRBG). Let \mathcal{D} be a λ -simple distribution sampler. Let p be the maximum number of random inputs I that are produced by \mathcal{D} . (This is implicitly also a restriction on \mathcal{A} calling REF.) l_i denotes the maximum block length of the i -th random input produced by \mathcal{D} . $L = \max\{l_1, \dots, l_p\}$ be the maximum block length of the random inputs, and let $\sigma = l_1 + \dots + l_p$ be their maximum total block length. We call a pair of blocks (a, b) , with $a, b \in \{0, 1\}^n$ a doubleblock. In particular, forkciphers take a (regular) block as input and output a doubleblock. Let B be the maximum number of doubleblocks requested from each individual next query (i.e. when l is the maximum number of requested bits, $B = \lceil l/2n \rceil$).

Theorem 2. *Let F be a forkcipher, that we model as an ideal forkcipher. Let Cond be a condenser that does not have access to F and let $\text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q', l')$ denote the maximum advantage against Cond of any adversary making at most q' queries, where l' is the maximum block length of the random inputs to Cond. Let the construction FCRNG-c be as described above. Let \mathcal{D} be a λ -simple distribution sampler and \mathcal{A} be an adversary attacking G whose accounting of queries is given above. Then*

$$\text{Adv}_{\text{FCRNG-c}, \lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D}) \leq \frac{2q(B+2)^2}{2^{2n+1}} + \frac{4q^2}{2^k} + 4 \sum_{j=1}^p \text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q, l_j)$$

Proof. Since our construction FCRNG is very similar to CTR_DRBG, our proof follows a similar argument to the CTR_DRBG proof in [17].

Setup. We consider computationally unbounded adversaries and hence, without loss of generality, assume that \mathcal{A} is deterministic. Let S_{ideal} and S_{real} be the systems that model the oracles accessed by \mathcal{A} in $\mathbf{G}_{G, \lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D})$ with challenge bit $b = 0$ and $b = 1$, respectively.

For this proof we will create a third system S_{hybrid} . It implements S_{real} , but when it is asked to run FCTR-c, instead of using the underlying forkcipher F , it produces uniformly random bitstring of length $2n$ (i.e. the output length of F). As a consequence the output of this modified FCTR-c is uniformly random. In order to prevent trivial attacks, S_{hybrid} will run the original FCTR-c when the min-entropy level c is lower than the threshold λ . Furthermore we update FCTR-c in both S_{real} and S_{hybrid} to maintain two ordered lists Keys and Queries. The resulting algorithms are shown in Fig. 8. We write Keys(S) and Queries(S) with $S \in \{S_{\text{real}}, S_{\text{hybrid}}\}$ to denote the corresponding list of system S .

Note that S_{hybrid} and S_{ideal} are not necessarily indistinguishable, even though it might seem that way at first glance. Indeed, S_{ideal} only idealizes the output of next through the RoR oracle. S_{hybrid} on the other hand also influences the other

algorithms of the PRNG as well, such as the `setup` algorithm. Since the adversary is able to get the current state using the `Get` oracle, the adversary might be able to distinguish the output of the real `setup` algorithm (in S_{ideal}) from the one in S_{hybrid} which has access to uniformly random strings from the modified `FCTR-c`.

FCTR-c $_K^V$ (M) in S_{real}	FCTR-c $_K^V$ (M) in S_{hybrid}
$O \leftarrow \epsilon$ // empty string	$O \leftarrow \epsilon$ // empty string
$\text{ctr} \leftarrow 0$	$\text{ctr} \leftarrow 0$
if $c \geq \lambda$ then $\text{Keys} \leftarrow \text{Keys} \cup \{K\}$	if $c \geq \lambda$ then $\text{Keys} \leftarrow \text{Keys} \cup \{K\}$
while $ O < M $ do	while $ O < M $ do
$P = F_K^{V,b}([\text{ctr}]_n)$	if $c \geq \lambda$ then
$O = O P$	$P \leftarrow_{\$} \{0, 1\}^{2n}$
$\text{ctr} \leftarrow \text{ctr} + 1$	$\text{Queries} \leftarrow \text{Queries} \cup \{(K, T, P)\}$
if $c \geq \lambda$ then	else
$\text{Queries} \leftarrow \text{Queries} \cup \{(K, T, P)\}$	$P = F_K^{V,b}([\text{ctr}]_n)$
return $M \oplus O[1 : M]$	$O = O P$
	$\text{ctr} \leftarrow \text{ctr} + 1$
	return $M \oplus O[1 : M]$

Fig. 8. Updated FCTR-c Algorithm in security proof of FCRNG-c

Proof Argument. This proof is structured into the following 4 steps.

1. There is an adversary \mathcal{A}^* s.t.

$$\Delta_{\mathcal{A}^*}(S_{\text{real}}, S_{\text{hybrid}}) = \Delta_{\mathcal{A}}(S_{\text{ideal}}, S_{\text{hybrid}})$$

- 2.

$$\Delta_{\mathcal{A}}(S_{\text{real}}, S_{\text{hybrid}}) \leq \frac{2q^2}{2^k} + 2 \sum_{j=1}^p \text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q, l_j) + \frac{q(B+2)^2}{2^{2n+1}}$$

- 3.

$$\Delta_{\mathcal{A}^*}(S_{\text{real}}, S_{\text{hybrid}}) \leq \frac{2q^2}{2^k} + 2 \sum_{j=1}^p \text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q, l_j) + \frac{q(B+2)^2}{2^{2n+1}}$$

- 4.

$$\text{Adv}_{\text{FCRNG-c}, \lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D}) \leq \Delta_{\mathcal{A}}(S_{\text{real}}, S_{\text{hybrid}}) + \Delta_{\mathcal{A}^*}(S_{\text{real}}, S_{\text{hybrid}}) \quad (1)$$

From the above arguments it follows that

$$\text{Adv}_{\text{FCRNG-c}, \lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D}) \leq \frac{2q(B+2)^2}{2^{2n+1}} + \frac{4q^2}{2^k} + 4 \sum_{j=1}^p \text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q, l_j)$$

Proof of Item 1. To see this, define \mathcal{A}^* as follows. \mathcal{A}^* runs \mathcal{A} and uses its own oracles to answer \mathcal{A} 's oracle queries. However when \mathcal{A} queries RoR, \mathcal{A}^* answers with a uniformly random string if $c \geq \lambda$. When \mathcal{A} outputs its final guess b' , \mathcal{A}^* also outputs b' . Note that if \mathcal{A}^* is in the real world then it perfectly simulates S_{ideal} for \mathcal{A} . On the other hand, if \mathcal{A}^* interacts with S_{hybrid} , then \mathcal{A}^* perfectly simulates S_{hybrid} for \mathcal{A} .

Proof of bound on $\Delta_{\mathcal{A}}(S_{\text{real}}, S_{\text{hybrid}})$ (Item 2): Defining bad transcripts. We will now prove Item 2 of the above enumeration by using the H-coefficient technique.

First of all we define the bad transcripts. A transcript is called *bad* if one of the following conditions happens:

1. The transcript contains a query of \mathcal{A} to F or F^{-1} using some key $K \in \text{Keys}(S)$. In other words \mathcal{A} was able to guess or derive K .
2. There are two identical keys in $\text{Keys}(S)$.
3. Queries contains a tuple (K, T, P) twice from the same call to FCTR- c (i.e. two different counter values resulted in the same ciphertext P). This can happen in S_{hybrid} but not in S_{real} .

Note that there is no Bad event that relates keys used in FCTR- c with keys used by Cond, since Cond is required to be independent of F .

If a transcript is not bad then we say that it is good. Let $\mathcal{T}_{\text{real}}$ and $\mathcal{T}_{\text{hybrid}}$ be the random variable of the transcript for S_{real} and S_{hybrid} respectively.

Proof of bound on $\Delta_{\mathcal{A}}(S_{\text{real}}, S_{\text{hybrid}})$: Probability of bad transcripts. We now give a bound on the chance that $\mathcal{T}_{\text{hybrid}}$ is bad. Let Bad_i be the event that $\mathcal{T}_{\text{hybrid}}$ violates the i -th condition. By the union bound,

$$\Pr[\mathcal{T}_{\text{hybrid}} \text{ is bad}] = \Pr[\text{Bad}_1 \cup \text{Bad}_2 \cup \text{Bad}_3] \leq \Pr[\text{Bad}_1] + \Pr[\text{Bad}_2] + \Pr[\text{Bad}_3]$$

We will start by giving a bound on $\Pr[\text{Bad}_1]$. The keys in $\text{Keys}(S_{\text{hybrid}})$ were put there during a call to FCTR. They can be categorized as follows:

- **Idealized Keys:** The key was picked uniformly at random. (This is the case if there was enough entropy c during the previous call to FCTR.)
- **Normal Keys:** This key is the result of

$$K_i \leftarrow \text{FCTR}_{K_{i-1}}^{V_{i-1}}[F](\text{Cond}(I))$$

(Indeed, all keys in $\text{Keys}(S_{\text{hybrid}})$ that are not idealized keys must have been derived as described. Since $K_i \in \text{Keys}(S_{\text{hybrid}})$, we know that $c \geq \lambda$ in the FCTR call that had K_i as key. Furthermore we know that in the FCTR call before that $c < \lambda$, since K_i is not uniformly random. Because of the fact that c increased, there must have been a call to REF and hence Cond.)

There can be at most q idealized keys, hence with each guess, the adversary has a probability of $q/2^k$ to guess a key. Furthermore the adversary guessed at most q times, leading to a probability of $q^2/2^k$ at best for the adversary to guess an idealized key.

For each $j \leq p$, let $\text{Hit}_1(j)$ be the event that the key derived from the random input I_j is a normal key, and causes Bad_1 to happen. From the union bound

$$\Pr[\text{Bad}_1] \leq \frac{q^2}{2^k} + \Pr[\text{Hit}_1(1) \cup \dots \cup \text{Hit}_1(p)] \leq \frac{q^2}{2^k} + \sum_{j=1}^p \Pr[\text{Hit}_1(j)]$$

We will now give a bound for all $\text{Hit}_1(j)$. We know K_j was derived during a REF query, which does not output any information to the adversary besides γ and z from \mathcal{D} . At that point the adversary has the following options for the next query: (a) corrupt the state using Get or Set, or (b) use REF or RoR. If (a) is used, then $c \leftarrow 0$. We can rule out this possibility since K_j is only added to Keys if $c \geq \lambda$. Attempting to use REF to increase c also overrides K_j before it is added to the list. In case (b), note that $c \geq \lambda$ during this next query since we only consider λ -simple distribution samplers. Hence K_j is not being used at all and immediately replaced with a new uniformly random key.

As a consequence the adversary only can only use (γ, z) to guess K_j . This also implies guessing the first block of K_j , which is exactly the setting of $\mathbf{G}_{\text{Cond}}^{1\text{-blk-guess}}(\mathcal{A}, S)$. Thus we have

$$\Pr[\text{Hit}_1(j)] \leq \text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q, l_j)$$

Therefore, by summing up over all $\text{Hit}_1(j)$ we derive

$$\Pr[\text{Bad}_1] \leq \frac{q^2}{2^k} + \sum_{j=1}^p \text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q, l_j)$$

We now bound the probability of Bad_2 . For any idealized key the probability to collide with any of the other q keys in the system is at most $q/2^k$. Since there are at most q such keys, the probability of this happening is at most $q^2/2^k$. On the other hand for any normal key, the probability that another normal key coincides is bounded by $\text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(\mathcal{A}, S)$, similar to the argument regarding Bad_1 . The only difference is that instead of the adversary directly guessing the outcome of $\text{Cond}(I)$, now the environment “guesses” the outcome. This results in the same bound;

$$\Pr[\text{Bad}_2] \leq \frac{q^2}{2^k} + \sum_{j=1}^p \text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q, l_j)$$

We now bound the probability of Bad_3 . Since the adversary has q queries and asks for at most B doubleblocks from next with each query, FCTR might produce up to $B + 2$ doubleblocks (also counting the blocks needed to set the next state). Note that each doubleblock leads to one forkcipher call, as it outputs two blocks. The situation resembles the classical birthday attack situation: Each call to FCTR, causes up to $B + 2$ independent, random doubleblocks (in $\mathbf{S}_{\text{hybrid}}$).

If any of these collide, Bad_3 occurs. Hence for a single FCTR call this can occur with the probability of the birthday bound, i.e. approximately $(B + 2)^2/2^{2n+1}$. In total this means

$$\Pr[\text{Bad}_3] \leq \frac{q(B + 2)^2}{2^{2n+1}}$$

Summing up we have

$$\Pr[\mathcal{T}_{\text{hybrid}} \text{ is bad}] \leq \frac{2q^2}{2^k} + 2 \sum_{j=1}^p \text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q, l_j) + \frac{q(B + 2)^2}{2^{2n+1}} \quad (2)$$

Proof of Bound on $\Delta_{\mathcal{A}}(\mathbf{S}_{\text{real}}, \mathbf{S}_{\text{hybrid}})$: Transcript ratio. Let τ be a good transcript s.t. $\Pr[\mathcal{T}_{\text{hybrid}} = \tau] \geq 0$. We now prove that

$$1 - \frac{\Pr[\mathcal{T}_{\text{real}} = \tau]}{\Pr[\mathcal{T}_{\text{hybrid}} = \tau]} \leq 0 \quad (3)$$

Since the adversary \mathcal{A} and the original game environment are deterministic, all randomness of the transcript is determined by the randomness of the distribution sampler \mathcal{D} , the random instantiation of F and the random values that $\mathbf{S}_{\text{hybrid}}$ produces in the modified FCTR-c instead of querying F .

Therefore we can specify the probabilities as follows:

$$\begin{aligned} \Pr[\mathcal{T}_{\text{real}} = \tau] &= \Pr[\text{Inputs}] \cdot \Pr[\text{Prim}] \cdot \Pr[\text{Coll}_{\text{real}}] \\ \Pr[\mathcal{T}_{\text{hybrid}} = \tau] &= \Pr[\text{Inputs}] \cdot \Pr[\text{Prim}] \cdot \Pr[\text{Coll}_{\text{hybrid}}] \end{aligned} \quad (4)$$

where

- **Inputs** denotes the event that the distribution sampler samples the same values as was done for τ .
- **Prim** denotes the event that the primitive F agrees with the result of any granted query (be it a direct query to F by the adversary or through the experiment), where the used key is not in **Keys**.
- **Coll_{real}** denotes the event that, in \mathbf{S}_{real} , the randomly chosen F complies with the queries in **Queries**.
- **Coll_{hybrid}** denotes the event that, in $\mathbf{S}_{\text{hybrid}}$, the randomly chosen values in the modified FCTR comply with the queries in **Queries**.

Note that indeed $\mathbf{S}_{\text{hybrid}}$ only behaves differently than \mathbf{S}_{real} for queries in **Queries** (i.e. queries with the key in **Keys**). From Eq. (4) follows

$$\frac{\Pr[\mathcal{T}_{\text{real}} = \tau]}{\Pr[\mathcal{T}_{\text{hybrid}} = \tau]} = \frac{\Pr[\text{Coll}_{\text{real}}]}{\Pr[\text{Coll}_{\text{hybrid}}]}$$

Let $Q = |\text{Queries}|$. In $\mathbf{S}_{\text{hybrid}}$ the relevant queries are answered in a uniformly (and independently) random way. Hence

$$\Pr[\text{Coll}_{\text{hybrid}}] = \frac{1}{(2^{2n})^Q}$$

Next we examine the situation in S_{real} . We bound the probability that the i -th element in Q is the same as what the randomly chosen F outputs. Due to the definition of a good transcript, we know that either the key or the input message (i.e. counter value) is different for each call. If the key is the same as for, say, t previous calls, then F will take the specific value with probability $\frac{1}{(2^{2n}-t)}$. Otherwise the output will take the specific value with probability $\frac{1}{(2^{2n})}$, which is smaller. Hence over all queries

$$\Pr[\text{Coll}_{\text{real}}] \leq \frac{1}{(2^{2n})Q}$$

This proves Eq. (3).

Wrapping it Up. Finally, from Lemma 1 together with Eqs. (2) and (3) follows

$$\Delta_{\mathcal{A}}(S_{\text{real}}, S_{\text{hybrid}}) \leq \frac{2q^2}{2^k} + 2 \sum_{j=1}^P \text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q, l_j)$$

Proof of Item 3. This follows immediately from Item 2, since the latter applies to all adversaries and hence it applies to adversary \mathcal{A}^* as well.

Proof of Item 4. By the triangle inequality,

$$\begin{aligned} \text{Adv}_{\text{FCRNG-c}, \lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D}) &= \Delta_{\mathcal{A}}(S_{\text{real}}, S_{\text{ideal}}) \\ &\leq \Delta_{\mathcal{A}}(S_{\text{real}}, S_{\text{hybrid}}) + \Delta_{\mathcal{A}}(S_{\text{hybrid}}, S_{\text{ideal}}) \\ &= \Delta_{\mathcal{A}}(S_{\text{real}}, S_{\text{hybrid}}) + \Delta_{\mathcal{A}^*}(S_{\text{real}}, S_{\text{hybrid}}) \end{aligned} \quad (5)$$

□

5.3 Security of FCRNG-T

Theorem 3. *Let F be a forkcipher, that we model as an ideal forkcipher. Let Cond be a condenser that does not have access to F and let $\text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q', l')$ denote the maximum advantage against Cond of any adversary making at most q' queries, where l' is the maximum block length of the random inputs to Cond. Let FCRNG-T be as described above. Let \mathcal{D} be a λ -simple distribution sampler and \mathcal{A} be an adversary attacking G whose accounting of queries is given above. Then*

$$\text{Adv}_{\text{FCRNG-T}, \lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D}) \leq \frac{4q^2}{2^k} + 4 \sum_{j=1}^P \text{Adv}_{\text{Cond}}^{1\text{-blk-guess}}(q, l_j)$$

We omit a detailed proof, as it follows the same argument as the proof of Theorem 2, with the exception that Bad_3 is removed.

Theorem 4. *Let F be a forkcipher, that we model as an ideal forkcipher. Let the constructions (FCRNG-c, FCTRCond) and (FCRNG-T, FCTRCond) be FCRNG-c and FCRNG-T, respectively, where Cond is instantiated with FCTRCond. Let \mathcal{D} be a λ -simple distribution sampler and \mathcal{A} be an adversary attacking G whose accounting of queries is given above. Then*

$$\begin{aligned} \text{Adv}_{(\text{FCRNG-c}, \text{FCTRCond}), \lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D}) &\leq \frac{2q(B+2)^2}{2^{2n+1}} + \frac{4q^2}{2^k} + \frac{12pq}{2^n} + \frac{4p\sqrt{q}}{2^{\lambda/2}} \\ \text{Adv}_{(\text{FCRNG-T}, \text{FCTRCond}), \lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D}) &\leq \frac{4q^2}{2^k} + \frac{12pq}{2^n} + \frac{4p\sqrt{q}}{2^{\lambda/2}} \end{aligned}$$

Proof Sketch. We omit the full proof as it simply follows the same strategy as Theorem 2 and Theorem 3. The only difference here is that instead of requiring some condenser Cond without access to F , we now fixed it to FCTRCond (which uses F). This is possible since we separated the domains, to be specific FCTRCond prepends each tweak with 1 and both FCTR variants prepend each tweak with 0. Thus, we arrive at the same bound here as for Theorem 3, and filled in the bound of Theorem 1.

Comparison to CTR_DRBG. Hoang and Shen [17] analyzed the security of the original CTR_DRBG, and called its internal condenser CtE. To slightly modify their notation, B' is the maximum number of output blocks in any next-query (similar to B , which denotes the number of doubleblocks in this work). s is the total block length of those outputs. They proved the following security bound for CTR_DRBG:

$$\begin{aligned} \text{Adv}_{G_{\text{CTR_DRBG}}, \lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D}) &\leq \frac{2(B'+3)(s+3p)}{2^n} + \frac{6q(q+1)}{2^k} + \frac{6p(q+1)}{2^n} + \\ &\quad \frac{12p\sqrt{q}}{2^{\lambda/2}} + \frac{48(\sqrt{q}+1) \cdot \sqrt{L+2} \cdot (\sigma+2p)}{2^n} \end{aligned}$$

First we compare FCRNG-T when used with CtE to CTR_DRBG.

Corollary 1. *In the same setting as Theorem 3, let (FCRNG-T, CtE) be the construction FCRNG-T with CtE being the instantiation of Cond. Then*

$$\text{Adv}_{(\text{FCRNG-T}, \text{CtE}), \lambda}^{\text{rob}}(\mathcal{A}, \mathcal{D}) \leq \frac{4q^2}{2^k} + \frac{4pq}{2^n} + \frac{4p\sqrt{q}}{2^{\lambda/2}} + \frac{32\sqrt{q(L+2)}(\sigma+2p)}{2^n}$$

Corollary 1 follows from Theorem 3 by instantiating Cond with CtE and applying the CtE security bound in Theorem 1 of Hoang and Shen [17], which remains the same when viewing CtE in $\mathbf{G}_{\text{Cond}}^{1\text{-blk-guess}}(\mathcal{A}, S)$. It shows that we were able to eliminate the security bound's dependency on the number of output pseudorandom blocks.

Now we compare FCRNG-T with FCTRCond (see Theorem 4) to CTR_DRBG. We were able to eliminate both the summand $\frac{2(B+3)(s+3p)}{2^n}$ and $\frac{48(\sqrt{q}+1) \cdot \sqrt{L+2} \cdot (\sigma+2p)}{2^n}$. This means the security of FCRNG-T with FCTRCond is not impacted by the length of the random inputs nor from the amount of requested output bits.

6 Discussion

We first give a performance estimation measured independently of the concrete implementation or hardware platform. Then we provide a C-implementation that we benchmark and compare with our estimation.

Efficiency Estimations. The full details are described in Sect. 7. We compare CTR_DRBG (instantiated with e.g. SKINNY-128-256 instead of AES) with FCRNG (using FCTRCond and e.g. ForkSkinny-128-256). We utilize the fact that ForkSkinny-128-256 is applying $f = 21 + 27 + 27 = 75$ SKINNY rounds. For comparison SKINNY-128-256 runs $e = 48$ rounds. FCRNG outperforms CTR_DRBG for all internal algorithms. The expected efficiency gain is more than 21% for next and 60% for setup and refresh.

Overall this means that FCRNG is expected to run 21% to 57% faster than CTR_DRBG with SKINNY, depending on how often the reseeding happens. We illustrate this in Fig. 9.

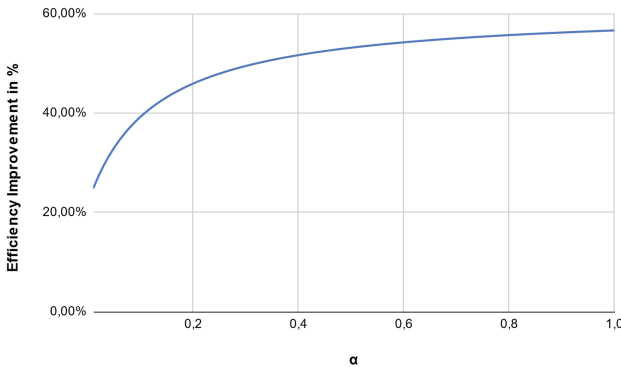


Fig. 9. Estimated efficiency gain of FCRNG with FCTRCond over CTR_DRBG. α denotes the ratio between the length of the random input data and output pseudo-random data.

Benchmark Setup. In order to validate our above estimations, we created a benchmark for the different PRNGs. The benchmarks were performed on a 64-bit machine with 4 CPUs (AMD EPYC 7713 64-Core Processor) and 4 GB of memory that runs Ubuntu 22.04 LTS. We implemented our construction FCRNG as well as CTR_DRBG, where the latter was implemented once with SKINNY as the internal cipher and once in its original form (i.e. with AES). The results are shown in Table 1.

We used the SKINNY implementation by Rhys Weatherley [29] and the AES implementation TinyAES [19]. For ForkSkinny we used an implementation by Erik Pohle [22] that is based on the previously mentioned SKINNY implemen-

Table 1. Runtimes of different PRNG implementations in CPU cycles.

	setup	refresh	next
FCRNG-T	43640	33609	881930
FCRNG-c	43640	33609	784252
CTR_DRBG (SKINNY)	105588	95557	987852
CTR_DRBG (AES)	153411	121998	1091351

tation. `setup` and `refresh` were called with 24 bytes of input, `next` was used to request 2000 bytes of pseudo-random output. The benchmarks were performed for ForkSkinny-128-256, SKINNY-128-256 and AES-128.

Benchmark Results. As shown in Table 1, the `refresh` function of FCRNG is the fastest by a large margin (65% faster than CTR_DRBG with SKINNY), as the previous efficiency estimations predicted. For `setup` the relative impact is slightly smaller (59%), since it performs some initializations. `next` of FCRNG-c is 21% faster than that of CTR_DRBG with SKINNY, as expected from the estimations of the last section. On the other hand `next` of FCRNG-T is only 11% faster than CTR_DRBG with SKINNY, even though it uses the same number of primitive rounds as FCRNG-c. This is due to the fact that here the key schedule has to be (partially) recomputed to accommodate for the tweak changes.

Overall FCRNG, especially FCRNG-c, outperforms the NIST standardized CTR_DRBG (i.e. using AES): `next` of FCRNG-c is 28% faster, `setup` and `refresh` are 72% faster. If in an exemplary scenario `refresh` is called once every 2000 bytes of requested `next` output, then FCRNG will be 33% faster.

7 Efficiency Estimations

For the estimation, we utilize the fact that ForkSkinny [2] uses the SKINNY round function. Therefore we can compare algorithms that use a forkcipher with ones that use a blockcipher. Hence we compare FCRNG (where ForkSkinny is used as the forkcipher F) with CTR_DRBG (where SKINNY is used as the blockcipher). We do not need to differentiate between FCRNG-c and FCRNG-T since they use the same number of primitive calls. We instantiate the condenser `Cond` in FCRNG with `FCTRCond`. Let f be the number of SKINNY rounds used in a specific ForkSkinny instance. Let e be the number of rounds in a comparable SKINNY instance.

- ForkSkinny-128-256 runs a total of $f = 21 + 27 + 27 = 75$ SKINNY rounds. For comparison SKINNY-128-256 runs $e = 48$ rounds.²

Below we give the round complexity of the internal algorithms used in CTR_DRBG and FCRNG.

² We look at the SKINNY instance with the same blocksize and tweak size, since we want the round function in both cases to operate on the same input sizes.

- CTR_DRBG related algorithms:
 - CtE[E, m] runs E a total of $(3\lceil(|I| + 64)/n\rceil + 6)$ times. This results in $3e^{\lceil(|I| + 64)/n\rceil + 6e}$ SKINNY rounds.
 - CTR runs E a total of $\lceil\frac{|M|}{n}\rceil$ times. This results in $e^{\lceil\frac{|M|}{n}\rceil}$ SKINNY rounds.
- FCRNG related algorithms:
 - FCTRCond runs F a total of $\lceil\frac{|I|}{k+n+v}\rceil$ times.
 - FCTR $_K^V(M)$ runs F a total of $\lceil\frac{|M|}{2n}\rceil$ times. This results in $f^{\lceil\frac{|M|}{2n}\rceil}$ SKINNY rounds.

First we focus on pseudorandom bit generation.

- next with r requested bits
(in CTR_DRBG:) runs CTR with $|M| = k + n + r$.
(in FCRNG:) runs FCTR with $|M| = k + n + r$ (since $w = n$ for our instances).
For simplicity, assume $|M|$ is a multiple of $2n$. Then the number of rounds in FCRNG divided by the number of rounds in CTR_DRBG is:

$$\frac{f^{\lceil\frac{|M|}{2n}\rceil}}{e^{\lceil\frac{|M|}{n}\rceil}} = \frac{f}{2e} = 75/96 \approx 0.78$$

This means we have an improvement of 22%.

We did the same efficiency estimation with ForkSkinny-64-192 (63 rounds) and SKINNY-64-192 (40 rounds). The result is an efficiency improvement of 21% for next. For ForkSkinny-128-384 (87 rounds) and SKINNY-128-384 (56 rounds) we reach an efficiency improvement 22% for next.

Now we will additionally consider the other two methods, i.e. **setup** and **refresh**, which are based on the condenser. Since we instantiated **Cond** in FCRNG with FCTRCond, we gain a significant efficiency improvement over the original CTR_DRBG (with CtE). The expected performance increase for **setup** and **refresh** is more than 60%. In order to give an efficiency comparison for the overall construction, we must make assumptions about how it is used. These are related to how often **refresh** and **next** are called, and how much random input data is used and how much pseudorandom data is requested. Also, the parameter v that is used in FCTRCond is relevant, as it controls how much data can be absorbed by a single call to the forkcipher F . However, in all settings we expect an improved performance. In order to capture different settings, we created the diagram in Fig. 9. It shows the efficiency gain of FCRNG with FCTRCond compared to CTR_DRBG. We chose $v = t/2$ and, in order to simplify calculations, assume **refresh** to be called once with $k + n + v$ bits of input data. The variable α in the diagram denotes the ratio between the length of the random input data fed to **refresh** and the length of the output pseudo-random data requested from **next**. Calculating the efficiency gain based on different values of α therefore gives an overview over different use cases, from **refresh** not being used at all to **refresh** being used for a lot of data. This means for $\alpha = 0.01$, we might call **refresh** once with 2 blocks of data and **next** with a total of 200 blocks. In that setting we have an efficiency

improvement of approximately 25%. Since we assume no application uses more random inputs than it requests pseudo-random outputs, we chose $\alpha = 1$ as maximum. With $\alpha = 1$ the setting is that `refresh` is called every time `next` is called, and the length of the random input is equal the amount of requested output bits. Here we have an efficiency improvement of approx. 57%.

8 Conclusion and Open Problems

In this work we presented our new PRNG `FCRNG`. It is similar to `CTR_DRBG` but optimized for lightweight applications through the use of a forkcipher and allowing lightweight primitives such as `SKINNY` and `ForkSkinny`. `FCRNG` is designed as a generic construction. It has two components, a counter-mode encryption `FCTR` and a condenser `Cond`.

For `FCTR` we propose the two instantiations `FCTR-c` (which is more performant) and `FCTR-T` (which is more secure, in particular when large chunks of pseudorandom data are requested at once). As for the condenser, while it could be instantiated with `CtE` (the condenser from `CTR_DRBG`, described by Hoang and Shen [17]), we also created a new forkcipher-based condenser `FCTRCond` that has better security and efficiency.

Our PRNG `FCRNG-T` (i.e. `FCRNG` with `FCTR-T`) has a better security bound than `CTR_DRBG`, since the security of `FCRNG` is not impacted by the length of the random inputs nor from the amount of requested output bits. `FCRNG-c` (`FCRNG` with `FCTR-c`) also has better security than `CTR_DRBG`, but its security does depend on the amount of requested output bits, similar to `CTR_DRBG`. Furthermore, as our benchmarks show, our constructions are faster than `CTR_DRBG`. `FCRNG-c` is faster by 28% or more, depending on how often one reseeds (i.e. calls `refresh`). The improved security and efficiency mean that `FCRNG` has benefits even when used in regular (non-lightweight) applications.

To prove the security of our construction we specified the equivalent of the ideal cipher model for forkciphers. We utilize the H-coefficient technique to prove our security bound.

As an open problem for future research is confirming the tightness of our security bound. Furthermore, novel forkciphers [3] could be used to improve this construction. Another direction is designing a multiforkcipher, which is a generalization of a forkcipher that was introduced in [1]. It can expand an input block to more than 2 output blocks. This can easily be integrated into the algorithms of this work (specifically `FCTR-c` and `FCTR-T`) to further optimize the efficiency.

A Security of `FCTRCond`

Algorithm Related Notation. Let `FCTRCond*` denote the algorithm that runs `FCTRCond` (as defined in Fig. 4) but only returns the first block, i.e. $\text{FCTRCond}^*(I) := \text{FCTRCond}(I)[1 : n]$. Let $M = \text{pad}^*(I_1)$, $M' = \text{pad}^*(I_2)$. According to the second line of `FCTRCond`, let M_1, \dots, M_a and M'_1, \dots, M'_b be the result of splitting M and M' , respectively, into blocks of length $k + n + v$. Let

B, B' be the output of $\text{FCTRCond}^*(I_1)$ and $\text{FCTRCond}^*(I_2)$, respectively. For all $i \in \{1, \dots, a\}$, let $X_i := F_K^{W,b}(V)[1 : n]$ with $K \leftarrow M_i[1 : k], V \leftarrow M_i[k+1 : k+n], W \leftarrow 1 \parallel [i]_{t-v-1} \parallel M_i[k+n+1 : k+n+v]$ (i.e. X is the i -th XOR-summand of B). We define Y_i accordingly: For all $i \in \{1, \dots, b\}$, let $Y_i := F_K^{W,b}(V)[1 : n]$ with $K \leftarrow M'_i[1 : k], V \leftarrow M'_i[k+1 : k+n], W \leftarrow 1 \parallel [i]_{t-v-1} \parallel M'_i[k+n+1 : k+n+v]$.

Simplifications. For the proofs in this section, we will only consider the case where $a = b$ (i.e. M and M' have the same length), as is stated in the following lemmas. The final goal is to give an upper bound on $\Pr[B = B']$. For any i , if $M_i = M'_i$ then $X_i = Y_i$, which means that X_i, Y_i have no influence on $\Pr[B = B']$, and could be removed. Since $I_1 \neq I_2$, there will be at least one index u at which $M_u \neq M'_u$. Hence, without loss of generality, we will always assume that $M_i \neq M'_i$ for all $i, 1 \leq i \leq a$.

Probability Theory. For a proper probability theoretic treatment, we define the event space $\Omega = \{(x_1, \dots, x_a, y_1, \dots, y_a) \mid \forall i : x_i, y_i \in \{0, 1\}^n\}$. In the tuple, the bitstring x_i should relate to the value of the random variable X_i (for y_i, Y_i accordingly).

Lemmas. As mentioned we assume that $M_i \neq M'_i$ for all $i, 1 \leq i \leq a$. Observe that the difference might either (a) cause a difference in the tweak or key portion of the F -call of X_i and Y_i , i.e. a difference in W or K , or (b) cause no difference in the tweak or key portion. In case (b) the difference must lie in the message portion V .

Lemma 5. *For any $i \in \{1, \dots, a\}$ and any $x, y \in \{0, 1\}^n, x \neq y$, If $M_i \neq M'_i$ then*

$$\Pr[X_i = Y_i = x] = \begin{cases} \frac{1}{2^n} \frac{1}{2^n} & \text{if case (a) applies} \\ 0 & \text{else (i.e. case (b) applies)} \end{cases} \quad (6)$$

$$\Pr[X_i = x, Y_i = y] = \begin{cases} \frac{1}{2^n} \frac{1}{2^n} & \text{if case (a) applies} \\ \frac{1}{2^n} \frac{1}{2^n - 1} & \text{else (i.e. case (b) applies)} \end{cases} \quad (7)$$

Proof. In case (a), X_i and Y_i were produced using different keys or tweaks, which means they are independent in the ideal forkcipher model. Hence the probability of any pair of values is $\frac{1}{2^n} \frac{1}{2^n}$. On the other hand in case (b), the F -calls were performed with the same tweak and key, but with different messages. Hence the outputs must be different, and there are in total $2^n(2^n - 1)$ possible values for the pair (X_i, Y_i) , each of which have the same probability. \square

Lemma 6. *For all $a', 1 \leq a' \leq a$, let $B_{a'} = X_1 \oplus \dots \oplus X_{a'}$, $B'_{a'} = Y_1 \oplus \dots \oplus Y_{a'}$. If $M_i \neq M'_i$, for all $i \in \{1, \dots, a\}$, then*

$$\forall a', 1 \leq a' \leq a, \forall d \in \{0, 1\}^n : \Pr[B_{a'} \oplus B'_{a'} = d] \leq \frac{1}{2^n - 1}$$

Proof. We will use induction on the number a' .

Induction base ($a' = 1$). We need to show that

$$\forall d \in \{0, 1\}^n : \Pr[X_1 \oplus Y_1 = d] \leq \frac{1}{2^n - 1}$$

Let d be arbitrary but fixed.

$$\Pr[X_1 \oplus Y_1 = d] = \sum_{x \in \{0,1\}^n} \Pr[X_1 = x, Y_1 = x \oplus d]$$

We can use Lemma 5 to bound the probability of $\Pr[X_1 = x, Y_1 = x \oplus d]$. In any case, $\Pr[X_1 = x, Y_1 = x \oplus d] \leq \frac{1}{2^n} \frac{1}{2^n - 1}$.

$$\Pr[X_1 \oplus Y_1 = d] \leq \sum_{x \in \{0,1\}^n} \frac{1}{2^n} \frac{1}{2^n - 1} = \frac{1}{2^n - 1}$$

Induction step. We assume as the induction hypothesis that

$$\forall d \in \{0, 1\}^n : \Pr[B_{a'} \oplus B'_{a'} = d] \leq \frac{1}{2^n - 1}$$

We need to prove that for $B_{a'+1} \oplus B'_{a'+1}$ the statement holds as well. Let $d \in \{0, 1\}^n$ be arbitrary but fixed. Let $E_{x,y}$ be a shorthand for the event $(X_{a'+1} = x, Y_{a'+1} = y)$.

$$\begin{aligned} & \Pr[B_{a'+1} \oplus B'_{a'+1} = d] \\ &= \sum_{x,y \in \{0,1\}^n} \Pr[B_{a'+1} \oplus B'_{a'+1} = d \mid E_{x,y}] \cdot \Pr[E_{x,y}] \\ &= \sum_{x,y \in \{0,1\}^n} \Pr[B_{a'} \oplus B'_{a'} = d \oplus x \oplus y] \cdot \Pr[E_{x,y}] \\ &= \sum_{x,y \in \{0,1\}^n} \frac{1}{2^n - 1} \cdot \Pr[E_{x,y}] \\ &= \frac{1}{2^n - 1} \sum_{x,y \in \{0,1\}^n} \Pr[E_{x,y}] \\ &= \frac{1}{2^n - 1} \end{aligned}$$

□

References

1. Andreeva, E., Bhati, A.S., Preneel, B., Vizár, D.: 1, 2, 3, fork: counter mode variants based on a generalized Forkcipher. *IACR Trans. Symm. Cryptol.* **2021**(3), 1–35 (2021). <https://doi.org/10.46586/tosc.v2021.i3.1-35>
2. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: Forkcipher: a new primitive for authenticated encryption of very short messages. In: Galbraith, S.D., Moriai, S. (eds.) *ASIACRYPT 2019*. LNCS, vol. 11922, pp. 153–182. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34621-8_6
3. Andreeva, E., Reyhanitabar, R., Varici, K., Vizár, D.: Forking a blockcipher for authenticated encryption of very short messages. *Cryptology ePrint Archive, Report 2018/916* (2018). <https://eprint.iacr.org/2018/916>
4. Ankele, R., et al.: Related-key impossible-differential attack on reduced-round SKINNY. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) *ACNS 2017*. LNCS, vol. 10355, pp. 208–228. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61204-1_11
5. Ankele, R., Kölbl, S.: Mind the gap - A closer look at the security of block ciphers against differential cryptanalysis. In: Cid, C., Jacobson Jr., M. (eds.) *Selected Areas in Cryptography – SAC 2018*. SAC 2018. LNCS, vol. 11349, pp. 163–190. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-10970-7_8
6. Barak, B., et al.: Leftover hash lemma, revisited. In: Rogaway, P. (ed.) *CRYPTO 2011*. LNCS, vol. 6841, pp. 1–20. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_1
7. Barak, B., Halevi, S.: A model and architecture for pseudo-random generation with applications to /dev/random. In: Atluri, V., Meadows, C., Juels, A. (eds.) *ACM CCS 2005*. pp. 203–212. ACM Press, Alexandria, Virginia, USA (7–11 November 2005). <https://doi.org/10.1145/1102120.1102148>
8. Beierle, C., et al.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) *CRYPTO 2016*. LNCS, vol. 9815, pp. 123–153. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_5
9. Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. *SIAM J. Comput.* **15**(2), 364–383 (1986)
10. Bogdanov, A., et al.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) *CHES 2007*. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_31
11. Chen, S., Steinberger, J.: Tight security bounds for key-alternating ciphers. In: Nguyen, P.Q., Oswald, E. (eds.) *EUROCRYPT 2014*. LNCS, vol. 8441, pp. 327–350. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_19
12. Cohnsey, S., et al.: Pseudorandom black swans: Cache attacks on ctr_drbg. In: *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1241–1258. IEEE (2020)
13. Desai, A., Hevia, A., Yin, Y.L.: A practice-oriented treatment of pseudorandom number generators. In: Knudsen, L.R. (ed.) *EUROCRYPT 2002*. LNCS, vol. 2332, pp. 368–383. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_24
14. Dodis, Y., Pointcheval, D., Ruhault, S., Vergnaud, D., Wichs, D.: Security analysis of pseudo-random number generators with input: /dev/random is not robust. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) *ACM CCS 2013*. pp. 647–658. ACM Press, Berlin, Germany (4–8 November 2013). <https://doi.org/10.1145/2508859.2516653>

15. Goldberg, I., Wagner, D.: Randomness and the Netscape browser. Dobb's J.-Softw. Tools Profess. Program. **21.1** 66–71 (1996). Redwood City, CA: M&T Pub., (1989–1996)
16. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED block cipher. In: Preneel and Takagi [23], pp. 326–341. https://doi.org/10.1007/978-3-642-23951-9_22
17. Hoang, V.T., Shen, Y.: Security analysis of NIST CTR-DRBG. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12170, pp. 218–247. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56784-2_8
18. Hoang, V.T., Tessaro, S.: Key-alternating ciphers and key-length extension: exact bounds and multi-user security. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 3–32. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_1
19. kokke: TinyAES. <https://github.com/kokke/tiny-AES-c>. Accessed 22 June 2022
20. NIST: NIST SP 800–22: Documentation and Software. <https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software>. Accessed 23 Nov 2022
21. Patarin, J.: The “Coefficients H” technique. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 328–345. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04159-4_21
22. Pohle, E.: ForkSkinny-C by Erik Pohle. <https://github.com/ErikP0/forkskinny-c>. Accessed 08 Sept 2022
23. Preneel, B., Takagi, T. (eds.): CHES 2011. LNCS, vol. 6917. Springer, Heidelberg (2011). <https://doi.org/10.1007/978-3-642-23951-9>
24. Raz, R., Reingold, O.: On recycling the randomness of states in space bounded computation. In: 31st ACM STOC, pp. 159–168. ACM Press, Atlanta, GA, USA (1–4 May 1999). <https://doi.org/10.1145/301250.301294>
25. Sadeghi, S., Mohammadi, T., Bagheri, N.: Cryptanalysis of reduced round SKINNY block cipher. IACR Trans. Symm. Cryptol. **2018**(3), 124–162 (2018). <https://doi.org/10.13154/tosc.v2018.i3.124-162>
26. Santha, M., Vazirani, U.V.: Generating quasi-random sequences from slightly-random sources (extended abstract). In: 25th FOCS, pp. 434–440. IEEE Computer Society Press, Singer Island, Florida (24–26 October 1984). <https://doi.org/10.1109/SFCS.1984.715945>
27. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: an ultra-lightweight blockcipher. In: Preneel and Takagi [23], pp. 342–357. https://doi.org/10.1007/978-3-642-23951-9_23
28. Tolba, M., Abdelkhalek, A., Youssef, A.M.: Impossible differential cryptanalysis of reduced-round SKINNY. In: Joye, M., Nitaj, A. (eds.) AFRICACRYPT 2017. LNCS, vol. 10239, pp. 117–134. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57339-7_7
29. Weatherley, R.: Skinny-C by Rhys Weatherley. <https://github.com/rweather/skinny-c>. Accessed 22 June 2022
30. Woodage, J., Shumow, D.: An analysis of NIST SP 800-90A. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 151–180. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_6
31. Zhang, P., Zhang, W.: Differential cryptanalysis on block cipher skinny with MILP program. Security and Communication Networks 2018 (2018)
32. Zhang, W., Rijmen, V.: Division cryptanalysis of block ciphers with a binary diffusion layer. IET Inf. Secur. **13**(2), 87–95 (2019)



DMA'n'Play: Practical Remote Attestation Based on Direct Memory Access

Sebastian Surminski¹(✉), Christian Niesler¹, Lucas Davi¹,
and Ahmad-Reza Sadeghi²

¹ University of Duisburg-Essen, Essen, Germany

{sebastian.surminski,christian.niesler,lucas.davi}@uni-due.de

² Technical University Darmstadt, Darmstadt, Germany

ahmad.sadeghi@trust.tu-darmstadt.de

Abstract. Remote attestation allows validating the trustworthiness of a remote device. Existing attestation schemes either require hardware changes, trusted computing components, or rely on strict timing constraints. In this paper, we present a novel remote attestation approach, called DMA'N'PLAY, that tackles these practical limitations by leveraging DMA (direct memory access). Since DMA does not require CPU time, DMA'N'PLAY even allows attestation of devices with real-time constraints. To prevent the exploitation of side-channels which potentially could determine if the attestation is running, we developed DMA'N'PLAY TO-GO, a small, mobile attestation device that can be plugged into the attested device. We evaluated DMA'N'PLAY on two real-world devices, namely a syringe pump and a drone. Our evaluation shows that DMA'N'PLAY adds negligible performance overhead and prevents data-only attacks, by validating critical data in memory.

1 Introduction

Embedded devices are crucial components deployed in smart factories, cars, medical devices, and critical infrastructures. They serve countless safety-critical tasks making their security of utmost importance. Despite their criticality embedded devices suffer from various security vulnerabilities [6, 26, 27, 68], including industrial robots [69], vehicles [49], and drones [7]. A well-known example of such a stealth attack on industrial control systems is Stuxnet [39], which targeted a uranium enrichment plant, remaining undetected for a long time and altering the configuration of centrifuges. Eventually, this attack not only physically damaged centrifuges in a long-term process by altering the power supply of the centrifuges [40], but also caused about 100,000 infections worldwide [52]. In a recent study, 48% of companies reported that they are unable to detect whether an IoT device on their network suffers from a breach, e.g., is part of a botnet [43]. This implies that attacks remain undetected for a long time and shows the urge for new security solutions to monitor critical IoT devices.

This problem is further amplified as embedded devices often lack basic security mechanisms that are common in most other types of systems [6]. For example, more than 80% of the embedded devices do not feature standard security mitigations such as ASLR, nonexecutable memory, and stack canaries [88].

Integrating new security mechanisms into such embedded devices is a challenging task: embedded devices are deeply integrated into other devices, hindering their replacement or the application of upgrades. Furthermore, these devices often have a long lifetime. Hence, there are many legacy devices operating for a long time. For instance, cars in the US are on average 12.1 years old [22], industrial robots have a lifetime of ten years [5, 17], and airplanes have a design lifetime of more than 30 years [4]. Both hardware and software of embedded devices are specifically tailored towards their use case. Furthermore, the hardware is deeply integrated into devices, such as machines, control units, and custom circuit boards (PCB). These circumstances hinder hardware replacements. Many of these devices also perform safety-critical or real-time tasks. During development, the correctness of functionality and timing behavior has been ensured [28, 86]. Altering such systems requires repeating these extensive testing routines.

Given these constraints, offloading detection of malware injection and data manipulation is the only feasible option. One way to do so is remote attestation as it allows an external entity to monitor and attest the internal behavior and state of a remote device [25]. Remote attestation enables a device, the so-called verifier, to check the integrity of another device, the prover. The main challenge in remote attestation is to perform trustworthy self-measurement of an untrusted device: even on a fully compromised system, an attacker may not be able to alter the attestation self-measurement. Many different remote attestation schemes have been proposed to tackle this problem [1, 2, 21, 23, 62, 74, 81]. These approaches have different complex demands that hinder their practical usage: hardware-based approaches require trusted computing modules like ARM TrustZone to perform secure measurements of the attested device [1, 2], which are often not available on small and embedded devices. Software-based approaches rely on precise measurements of execution time and therefore have strict requirements toward their implementation and communication, limiting their practical applicability [23, 74, 81]. Hybrid approaches need custom hardware extensions, that are expensive for initial implementation, and are not available on legacy devices [21, 62].

Contributions. In this paper, we present DMA'N'PLAY, a new remote attestation framework that leverages DMA (Direct Memory Access) to observe the operation of embedded devices. It allows the integrity of the device to be verified during operation without requiring any trusted computing modules, hardware modification, or changes to the software of embedded applications. The general idea of DMA'N'PLAY is to enable the verifier to directly monitor the memory of the attested device using DMA. In traditional attestation schemes, trusted computing components or custom hardware extensions are used to perform a secure self-measurement. However, in case of legacy embedded systems that do not feature such components, integrating these attestation schemes implies replacing the components of the embedded system, a costly and impractical process, as

eluded earlier. Here, DMA’N’PLAY is a viable solution: Instead of replacing the hardware components, we add a tiny and low-cost device to perform the attestation. In addition, this also has a second advantage: DMA allows direct access to a system’s memory without the involvement of the processor. DMA is typically used to speed up memory access of external devices and reduce the utilization of the processor. As the DMA controller is independent of the attested device, DMA enables trustworthy self-reports even on compromised systems. It is a standard feature of microcontrollers and is widely available in embedded devices as used in industry; supported by all major vendors such as STMicroelectronics [79], NXP [66], and Infineon [45].

In contrast to traditional pure software-based attestation [74], integrating the DMA’N’PLAY framework into existing applications is straightforward, as no complex runtime requirements have to be considered and no extensive execution time thresholds have to be provided. Moreover, DMA’N’PLAY is also suitable for timing-critical devices, e.g., real-time or medical devices, where any change in hardware or software implies re-validation of the timing behavior. The implementation of DMA’N’PLAY does not change the software on the attested device. This makes the DMA’N’PLAY attestation framework also suitable for legacy devices as neither hardware modifications nor source code of the attested devices are required.

In DMA’N’PLAY, we use DMA to give an external device direct access to the main memory. This way, the external device can examine the main memory of another device during run-time. Using this direct access to main memory, the actual data in memory can be monitored, e.g., variables and data structures. This enables DMA’N’PLAY to detect malicious manipulations on data in memory, detecting data-only attacks that cannot be covered by traditional security techniques like control-flow integrity. We present a format for configuration that allows specifying the data structures to monitor and define constraints for valid states. The attested device cannot influence this investigation, as the memory access is completely handled by the DMA controller. In contrast to traditional remote attestation schemes, DMA’N’PLAY requires the verifier to be directly connected to the attested device. In practice, depending on the setting, the verifier can either be a standard computer system, e.g., a personal computer, smartphone, tablet, or an embedded device like a diagnostics terminal in a repair workshop. Additionally, we propose a tiny embedded device, dubbed DMA’N’PLAY TO-GO, that can be used to relay the attestation measurements to a remote verifier or that can also be used directly as a verifier. For instance, DMA’N’PLAY TO-GO can forward attestation measurements to an external verifier, for example via a wireless transmission. This way, also mobile devices like drones or vehicles can be attested during operation.

In summary, we provide the following contributions:

- We propose DMA’N’PLAY, a novel remote attestation framework that uses DMA to monitor the attested device and specifically the content of its memory, thereby detecting manipulations and illegal states without requiring

changes to either hardware or software, e.g., reprogramming or instrumentation of the attested device.

- We show how the DMA'N'PLAY attestation framework uses a binary file of the attested device and a configuration file to define benign states of the attested device, allowing integration into both new and existing legacy devices.
- With DMA'N'PLAY TO-GO, we present an external verification device that can be attached to attested devices to continuously check their integrity.
- We provide integration guidelines and show the necessary steps to implement attestation into devices.
- To show its applicability, we integrated DMA'N'PLAY into two real-world systems, a medical device, and a drone. In a case study, we use DMA'N'PLAY to detect attacks on these devices: a manipulation of the injection rate of a syringe pump and modifications to the drone control system.

2 Background

In this section, we explain direct memory access (DMA) and give background on standard serial communications protocols used for DMA'N'PLAY.

2.1 U(S)ART and SPI

Microcontrollers usually interact with external peripherals such as sensors, displays, and control units. Hence, they need standardized interfaces such as UART (universal asynchronous receiver-transmitter) [54] and SPI (Serial Peripheral Interface) [29] to exchange data. The UART interface has two communication lines: one for transmitting data (TX) and one for receiving data (RX). UART is asynchronous, i.e., it operates on a fixed clock cycle, which the receiver needs to be aware of to correctly interpret the data. USART (Universal synchronous and asynchronous receiver-transmitter) offers an additional clock signal, which is used to synchronize the transmitter and receiver. UART and USART communication is designed for direct bilateral device communication [29]. In contrast, SPI is designed for the communication of one single device (master) to multiple peripherals (slaves) like displays or sensors. SPI allows full duplex and synchronous communication using four lines: a serial clock provided by the master, data output from the master (MOSI), data output from the slave (MISO), and slave select (\overline{SS}) to indicate the master and slave configuration [61].

2.2 Direct-Memory-Access (DMA)

Direct-Memory-Access (DMA) is a common feature of microcontrollers that allows to directly copy memory contents from or to external peripherals. With DMA, the CPU does not need to manage the copying of data between RAM and peripherals. The purpose of DMA is to unburden the processor of the time-consuming task of moving data over the memory bus between places. The CPU

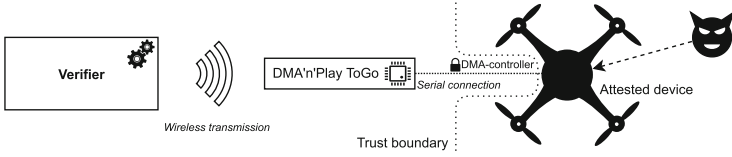


Fig. 1. The adversary can fully compromise the attested device, but cannot attack the verifier.

does not need to actively poll for incoming or outgoing data transfers, and the CPU is not frequently interrupted (i.e., by CPU interrupts) to move small bits of data [67]. DMA is a common and widespread feature on standard micro-controllers as used in the industry, supported by all major vendors including STMicroelectronics [79], NXP [66], and Infineon [45]. The DMA module can be configured to directly copy data from or to an external peripheral like UART. The CPU now only needs to be notified, e.g., by an interrupt once the copy routine has terminated. Note that a memory controller typically features multiple parallel DMA streams and offers precise predictability of execution times for real-time applications.

3 Assumptions and Attacker Model

Figure 1 shows our threat model and trust assumptions. The attacker can fully compromise the attested device. However, the attacker cannot compromise the external verifier or the DMA’N’PLAY TO-GO and cannot alter the configuration of the DMA controller.

Assumptions. We assume an embedded device that features a DMA controller that allows copying memory content to an external bus, e.g., a serial bus like UART or SPI. This is a common feature of DMA controllers deployed on different embedded devices [45, 66, 79]. Furthermore, we assume that the configuration of the DMA controller can be locked, e.g., by utilizing a memory protection unit (MPU) preventing the DMA configuration from being changed even when the entire system is compromised. This is also a widespread feature on existing micro-controllers [46, 47, 77]. We describe in detail how this can be achieved in Sect. 5.4. Second, as in any remote attestation scheme, we require a trusted verifier. During attestation, the verifier must be attached to the serial bus. The verifier device does not have to be connected all the time but is only required during attestation time. The role of the verifier can be taken over by a commodity computer system, e.g., a notebook or workstation. The verifier can also be integrated into a diagnostics system typically used for repair and maintenance in the automobile space. Alternatively, we developed DMA’N’PLAY TO-GO, a dedicated, small, low-cost embedded device to take over the role of the verifier and either perform the verification directly or relay the information to a remote verifier. Figure 1 shows this scenario: DMA’N’PLAY TO-GO is directly attached to the attested device and transmits the

measurements to the external verifier via a wireless communication channel. This communication can be encrypted, for example using TLS (Transport Layer Security). Both devices, the verifier and DMA'N'PLAY TO-GO are trusted. The third requirement is the knowledge of the firmware of the attested device and its benign states. For this, the verifier needs to access the firmware in the ELF (Executable and Linkable Format) binary format. Note that DMA'N'PLAY does not require source code. When compiled with debugging symbols, the verifier can identify the addresses of variables and data in memory by their names in the source code. To check the content of variables and identify illegal states, the verifier requires information about benign states, e.g., valid ranges of variables. We provide a configuration file format in which this information can be provided along with integration guidelines that describe how to integrate DMA'N'PLAY into new or existing applications, see Sect. 6.4.

Attacker Model. We assume a remote adversary. The adversary can compromise the attested device at any point in time and is able to modify program data or configuration data, e.g., by means of typical software vulnerabilities like memory errors or insecure or insufficiently protected interfaces. However, high-privileged operations like changes to the MPU (Memory Protection Unit) are not possible. In Sect. 5.4 we show how this can be achieved on standard commodity microcontrollers.

Similar to other remote attestation approaches, we exclude physical attacks on the devices [1, 2, 62, 74]. Thus, the attested device and particularly its hardware cannot be tampered with, including the serial connection between the prover and the external verification device. Furthermore, also along with other remote attestation approaches, we assume that attacks on the verifier are out of scope.

4 Challenges

Assuring the integrity of a remote system is a challenging task. The attested system itself is untrusted, so obtaining trustworthy reports from such a system is a complex problem.

In particular, a secure attestation scheme needs to tackle the following challenges:

Challenge 1: Secure Self-measurement. The attestation scheme must guarantee that the attested device is not able to manipulate or delay the self-measurement.

Challenge 2: Detectability. When using an attestation approach that is not constantly running it is important that the attacker cannot detect whether the device is currently being attested. Otherwise, the attacker could hide or stop any attacks while the attestation is being executed. This includes both direct observations as well as side-channels, e.g., monitoring other events that indicate an attestation.

Challenge 3: Root of Trust. In software-based remote attestation, any secret key on the attested device can be obtained upon full system compromise.

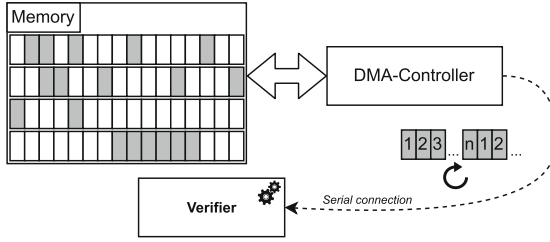


Fig. 2. The DMA controller sends all relevant memory sections to the verifier in a circular process.

The lack of a root of trust results in the problem, that the verifier cannot distinguish between the genuine device, other devices, or simulations.

Challenge 4: Side-Effects and Real-Time Operation. Attestation may not negatively influence the normal operation of the attested system, especially when performing tasks with real-time constraints. This is especially important when integrating attestation into existing legacy devices, whose timing behavior has already been validated and hence may not be altered.

Challenge 5: Efficient Verification. Implementing an efficient and effective verifier is a challenging task. Typical attestation protocols send hashes of the system’s current state, from which the actual state cannot be reconstructed directly. The verifier requires a full lookup table of all benign states. Generating such a table is a complex task and requires a large amount of memory.

In the following, we will show how DMA’N’PLAY allows remote attestation while addressing these challenges by utilizing a standard DMA interface.

5 Concept

Our concept of DMA’N’PLAY is to use Direct Memory Access (DMA) to enable an external device, called verifier, to observe the memory of the attested device. Figure 2 shows the high-level idea of our approach. In an infinite loop, the DMA controller sends relevant memory content to the verifier, allowing the verifier to monitor memory contents such as configuration data, measurements, and other static and variable memory content. We use a one-way serial connection to send the data of the attested device to the verifier, so there are no interdependencies between these devices. Furthermore, since there is no feedback from the verifier, the attested device cannot determine whether the verifier device is present.

5.1 Using DMA for Attestation

For DMA’N’PLAY, we configure the DMA controller in such a way that it shifts memory contents for attestation to an external peripheral via a serial connection, e.g., UART. The verifier receives the raw memory contents from the attested device

and verifies its integrity. Serial communications like UART use two separate lines for sending and receiving (see Sect. 2.1). By only connecting the pins for sending on the attested device with the receiving pin of the verifier, a one-way transmission is ensured. This gives two security benefits in contrast to traditional attestation schemes. First, the attested device does not get any feedback from the verifier. Hence, the attested device cannot determine whether it is currently being attested. Second, in case of implementation flaws in the verifier, the attested device faces significant limitations to exploit these flaws as there is no feedback from the verifier.

DMA tasks are usually configured once on set up and are typically not required to change during run-time. We set up the DMA controller to frequently push memory contents (SRAM, configuration data, content of variables) over UART for attestation. Since DMA does not need to be reconfigured, access to the DMA controller can be blocked by the memory protection unit (MPU). Consequently, the external device will always receive unaltered memory contents. As the DMA controller is independent of the processor and the software operating on the device, this does not influence the normal operation of the attested system. This makes DMA'n'PLAY also suitable for attesting devices with real-time requirements, i.e., where the correct operation also requires maintaining strict timing thresholds.

5.2 DMA'n'PLAY Attestation

The DMA'n'Play attestation scheme takes advantage of its full memory access to ensure run-time constraints on specific variables. This is unlike traditional attestation schemes, in which memory is being hashed and then sent to the verifier for verification, which makes reconstructing the original content a challenging and complex task. For instance, in these schemes, the verifier compares the hash values to a list of known hash values of benign states, requiring a database of hash values of all valid states. This approach has several drawbacks. It requires a pre-computation of valid states, an extensive task, leading to the well-known state explosion problem [85]. Changes to the attested device require updates of these states, even upon small changes such as modifications of individual parameters or updates to the attested application. Furthermore, the database of valid states requires significant memory on the verifier's side. On the contrary, in DMA'n'PLAY attestation, the verifier has access to the raw data in the memory of the attested device. By mapping the memory content to relevant information like data or control variables, the verifier can monitor the attested device's internal state. With DMA'n'PLAY, we recommend a model-based approach to avoid the state explosion. That is, the behavior of a model of the attested device is compared to its actual behavior. Direct access to the raw memory content allows the verifier to perform complex checks on state data, e.g., assuring that variables are within specific ranges or validating specific dependencies between variables.

To attest a device, the verifier solely requires the binary of the attested device and a configuration file that specifies benign states (see Sect. 6.2). If the attested device is modified or updated, only the binary has to be replaced. If the configuration changes, only the rules in the configuration file have to be adapted. This allows simple updates of the attestation rules, when the attested system changes,

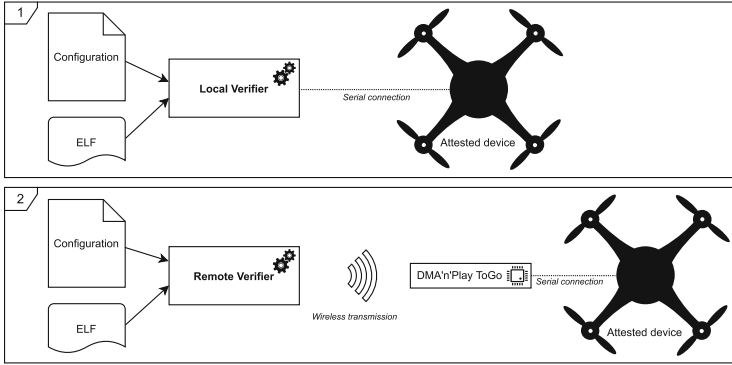


Fig. 3. The verifier uses the compiled binary and a configuration file to attest the device. The verifier can either be directly connected to the attested device (Case 1), or communicate via DMA’N’PLAY TO-GO (Case 2).

for example, due to a new software version. By mapping the memory content to relevant information like sensor or control variables, the verifier can reconstruct the state and behavior of the attested device. Relations between sensor information and output variables allow detecting compromises and manipulations, creating a deviation between the actual and expected behavior. However, creating such a behavior model is challenging without source code and deep knowledge of the device. Reverse engineering is helpful, but often requires significant effort as states and interdependencies are hard to reconstruct.

However, when the binary is enhanced with debugging symbols or even the full source code is available, the verifier can identify variables and data structures by their names. With DMA’N’PLAY and application knowledge, it is then also possible to implement sophisticated verification logic and rules for device behavior. Furthermore, the verifier can also monitor changes over time or access sensor input. In embedded devices, communication with external peripherals, such as sensors, typically takes place through special memory areas, which can also be covered by DMA’N’PLAY. In addition, the verifier can use sensor information from other attested devices, e.g., drones in a swarm, and compare it with the currently attested device. Summing up, the complete availability of the attested system’s memory and unrestricted access to it allow straightforward implementation of checks of the attested system to verify its integrity and detect manipulations. While DMA’N’PLAY does not necessarily require the source code of the attested system, developing sophisticated rules for the verifier requires insight into the precise functionality of the attested system, which is typically only given via source code.

5.3 Verifier

The verifier checks the correctness of the attested device based on the raw memory content data provided via DMA. This data is automatically sent in a circular

process by the memory controller of the attested device. For the attestation, the verifier needs to interpret these raw values. To do so, the verifier takes the compiled binary ELF file and analyzes it to obtain the memory regions to be attested. Note that DMA'N'PLAY does not need the source code of the firmware of the attested device. Embedded devices typically have a static memory configuration. Therefore, the exact memory layout can be initially determined using the binary firmware. If the binary is compiled with debugging symbols, the verifier can identify variables and data in memory by their respective names, find their location in the data stream, and reconstruct the content of the memory in the attested device. This makes it possible to perform complex checks on the memory of the attested device. DMA'N'PLAY requires the verifier device to be close to the attested device due to the communication channel. The verifier can be implemented on any commodity computer system as long as it can be equipped with a serial interface, e.g., personal computers, mobile devices such as smartphones or tablets, or specialized systems like a diagnostics terminal for cars or planes.

For the attested device, it does not matter whether a verifier is attached or not. As there is no input from the receiving verifier device, the attested device is unable to ascertain whether a verifier is present. Therefore, the attacker cannot determine whether the device is being attested or identify the memory locations that are currently being transmitted. However, an attacker could use other information on the device to determine if it is likely that the device is being attested: For example, in the case of a plane or a car, if it is flying or driving, it is unlikely that an external verifier integrated in a diagnostics system that is usually used in a garage is attached. To counter this, we developed a verification solution called DMA'N'PLAY TO-GO that can be integrated into other devices to continuously attest devices also during operation. DMA'N'PLAY TO-GO is a small embedded device that is being connected to the serial interface and can either directly perform the verifier task or relay the data via a wireless interface, e.g., Bluetooth or Wi-Fi, to a remote verifier. In practice, DMA'N'PLAY TO-GO will be used to forward the attestation measurement to an external verifier, as this allows the integration of more complex attestation tasks and also the usage of configuration files for verifier, which a small embedded device is not able to process. Figure 3 shows the general architecture of DMA'N'PLAY and its two operating modes: The verifier can either be directly connected to the attested device (Case 1) or communicate via DMA'N'PLAY TO-GO (Case 2). In the latter case, DMA'N'PLAY TO-GO is directly connected to the attested device and relays the data to the verifier, e.g., over Wi-Fi or Bluetooth. The attested device in Fig. 3 is represented by a drone. In Sect. 7.1 we provide a case study on a syringe pump and a drone.

5.4 Locking of DMA-Controllers

The DMA controller sends the content of the attested memory to the verifier. The security of DMA'N'PLAY attestation is based on the assumption that the attacker cannot change the configuration of the DMA controller. Otherwise, the

attacker could alter the DMA controller such that critical memory areas are not being reported, thereby hiding modifications and attacks.

Embedded devices in general feature a memory controller, which allows restricting access to arbitrary memory regions. The most basic form of such a controller is the Memory Protection Unit (MPU). It is a common and widespread feature on standard controllers [46, 47, 77]. A properly configured MPU will define protected memory areas and block unprivileged access. It is possible to lock the DMA configuration, by restricting unprivileged access to the memory section that contains it.

Devices with an MPU should provide at least two privilege modes. The basic configuration has one privileged mode with access to all resources, and one unprivileged mode with limited capability. In order to ensure the memory access rights, the code is run either in privileged or unprivileged mode. Within the unprivileged mode, all memory restrictions defined in the MPU are strictly enforced. Once the processor operates in unprivileged mode, switching back to privileged access is only possible through a Super Visor Call (SVC). This triggers an interrupt that checks the legitimacy of the request and either allows or denies the mode switch. Since MPU restrictions are only enforced in unprivileged modes, it is important to avoid critical bugs in privileged code. Therefore, firmware analysis [58] and fuzzing [41] are performed. Furthermore, the amount of privileged code is minimized by separating tasks based on their required permission level. This is often implemented in software, e.g., by the operating system of the microcontroller. TockOS [55], for example, divides the OS kernel into a trusted core for critical tasks and untrusted capsules for peripheral drivers and other noncritical tasks. EPOXY [24] uses the MPU to provide two domains and requires manual annotations by the developer. Sometimes ISA properties, such as unprivileged memory instructions, are used to enable execute-only memory protection schemes (uXOM [51]).

While privilege separation has been neglected in the past it is now a critical task for the software developer. Recently, frameworks such as EPOXY [24] have emerged in academic research, applying a technique called privilege overlaying to only execute the necessary operations in privileged mode. This considerably advances the protection of hardware configurations including the DMA controller. Some frameworks such as D-Box [59] explicitly address the topic of DMA locking and DMA security. D-Box [59] allows the compartmentalization of the DMA controller on embedded devices like ARMv7-M boards, using a software reference architecture and the capability of the MPU. Thus, the DMA configuration can be sufficiently protected from a potential attacker. EPOXY [24] and D-Box [59] enable us to provide a secure channel over DMA to the external verifier. Thus, we are able to add remote attestation to devices, that had no feasible attestation option until now (either due to hardware or system limitations).

Some microcontrollers even feature more sophisticated protection solutions, such as a complete memory management unit (MMU), e.g., the ARM Cortex-A family [13], up to a full trusted execution environment (TEE) such as TrustZone. TEEs guarantee authenticity of the executed code, integrity of runtime states,

and confidentiality of code and data [73]. For example, TrustZone is an optional, but common extension of the new and more sophisticated ARMv8-M [12] and ARMv8-A [14] microcontroller architecture. Unfortunately, as of now, TrustZone has very limited availability on existing microcontrollers [59]. In the RISC-V architecture such memory restrictions are enabled using the Physical Memory Protection (PMP) features included in the RISC-V instruction set [71]. Furthermore, for RISC-V there exist TEE solutions such as Keystone [53]. However, as of now, Keystone requires the privilege modes S, U, and M [53]. Unfortunately, these privilege modes are also optional. For example, the new and popular ESP32-C3 only implements the unprivileged U (user) and privileged M (machine) mode [37].

5.5 Hardware Requirements and Target Platforms

In summary, DMA'N'PLAY requires three hardware properties: (1) A DMA controller with a peripheral such as SPI or UART to directly output memory contents to the external attestation device. (2) An MPU, which locks the DMA configuration. (3) Privilege separation with at least two modes to prevent reconfiguration of the DMA controller by the attacker.

These requirements are fulfilled by numerous hardware platforms. We focus on the ARMv7-M architecture, which we also use in our case study in Sect. 7.1. This architecture is in widespread use in the industry with many legacy devices. The successor ARMv8-M is also suitable for DMA'N'PLAY. It even has optional support for TrustZone [73]. However, TrustZone support is purely optional and there will be new boards without TEE. Recently, the RISC-V architecture became popular, especially in the embedded domain. As discussed in Sect. 5.4, DMA'N'PLAY can also be used on RISC-V devices.

5.6 Devices Without Source Code

The DMA'N'PLAY framework can also be used on applications where there is no source code, but only the compiled binary is available. For example, the source code of legacy devices is often either missing, incomplete, or unavailable. In particular, there is a huge amount of legacy devices in machinery designed for a long service life, e.g., in power plants, factories, professional equipment, and vehicles. In these environments, there are often embedded devices with discontinued software support, leading to critical security risks [49, 69]. If no source code is available, DMA'N'PLAY can be integrated with the help of binary rewriting. Binary rewriting describes the modification of a given compiled and possibly (dynamically) linked binary in such a way that it remains executable [87]. Binary rewriting can either be done dynamically (during execution) or statically (on a binary that is not currently being executed) [87]. Due to the added complexity of run-time execution, dynamic rewriting is more challenging than static rewriting. In static rewriting, all binary transformation steps can be executed in a row, while dynamic rewriting requires an iterative algorithm [87]. Furthermore, a persistent dynamic binary transformation will induce time and memory overhead

during run-time [87]. For most IoT devices static rewriting will be sufficient to integrate DMA’N’PLAY, as devices can be usually re-flashed with a new binary during maintenance.

In order to integrate DMA’N’PLAY into a binary the following steps are required: (1) Integrate a DMA configuration into the binary, to output the entire memory through DMA. (2) Integrate an MPU-based lock. (3) Set up the verifier with information on the memory content and data of interest for the attestation process. Depending on the amount of information available regarding the memory structure and the variables of interest, the verifier can be configured appropriately. The integration guidelines described in Sect. 6.4 apply. The only difference is that the DMA output and MPU lock on the attested device are integrated and configured with the help of binary rewriting techniques, rather than directly added to the source code and recompiled.

The integration of the DMA output (Step 1) and MPU-based locking (Step 2) are straightforward on the binary level. Both steps represent a reconfiguration of the hardware. On the software side, this reconfiguration is equivalent to privileged write operations on registers and memory positions (Memory Mapped I/O [70]). DMA’N’PLAY performs this configuration step once during device startup and drops all privileges afterward. Thus, steps (1) and (2) can be achieved by adding a static code block to the part of the binary executed at the end of the device startup.

6 Implementation

To show the applicability of DMA’N’PLAY for different computing architectures, we integrated DMA’N’PLAY into a syringe pump and a drone. A syringe pump is a medical device that automatically injects medicine into a patient’s body. Drones are manually controlled or autonomous flying devices. First, the attested device (syringe pump or drone) needs to be modified such that its memory contents are sent to the verifier. Second, the verifier has to be provided with information on the benign states of the attested device. Next, the verifier needs to check the validity of the received data and report manipulations. In Sect. 6.4 we provide detailed integration guidelines that describe how DMA’N’PLAY can be integrated into devices.

6.1 Attested Device

To implement DMA’N’PLAY on the attested device, we first determine the relevant memory content. These memory contents typically include variables that represent the state and configuration of the device. To reduce the amount of transmitted data, we modify the linker file to create a dedicated memory section containing all the data to be included in the attestation. We call this section the *attestation section*. This is an optional step, it is also possible to attest the complete memory of the attested device. We use the built-in source code annotation capability of the GCC compiler, called attributes [44], to assign variables

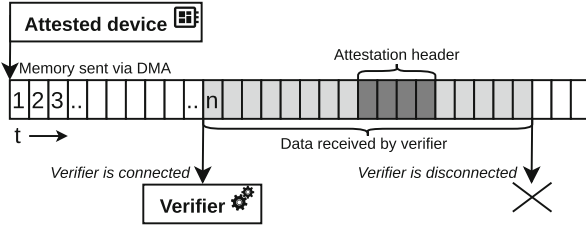


Fig. 4. The verifier uses the attestation header to identify the position in the data stream.

to the *attestation section*. Such functionality is also available in other compilers such as CLANG [84]. In Sect. 5.1, we briefly describe the DMA feature that we use to transfer memory content to an external device. We configure the DMA controller on the chip to output the contents of the attestation section over a peripheral interface. We set the DMA controller to circular mode and configure a direct DMA stream from memory to peripheral. Thus, the attested device sends the memory content in an endless loop.

To enable the verifier to determine the current position in memory, the attested memory features a so-called attestation header, which is a static string at a known position in the attested memory. Using the attestation header as a reference point, the content of all variables and data structures in the attestation section can be identified. This is especially relevant as the verifier can be attached at any time. Thus, the data stream of the attested device can be at any random position. Figure 4 illustrates this scenario.

To secure the DMA controller from malicious access, we utilize the Memory Protection Unit (MPU) as described in Sect. 5.4. Since the configuration of the DMA is handled through memory-mapped I/O, we restrict access to the memory area that contains the DMA configuration. We drop the privilege level that is required to reconfigure DMA after startup. Thus, a remote attacker cannot alter or influence the data transmitted to the external verifier via DMA. Even if the remote attacker gains arbitrary code execution on the device, the attacker is still missing the required privilege level to alter the DMA configuration.

Typically, the source code of the attested device is available and can be recompiled with our modifications, i.e., the dedicated attestation memory section, modified DMA configuration, and MPU-based lock. However, for some legacy devices source code is not available for various reasons. In case the source code is not available, we output the entire memory content for attestation and leverage binary rewriting techniques. We provide further details on this in Sect. 5.6. In the integration guidelines in Sect. 6.4 we explain all the steps necessary to integrate DMA'N'PLAY.

6.2 Verifier

The verifier receives and validates the data from the attested device. To do so, the verifier requires an ELF file of the software running on the attested device.

With this file, the verifier automatically reconstructs the memory layout of the attested device, i.e., the locations of variables and data structures. Using this information, the verifier determines the location and values of these variables in the data stream from the attested device. When compiled with debugging symbols, the verifier can identify the addresses of variables and data in memory by their names in the source code. Otherwise, manual mapping of variable names to memory addresses is required. We implemented the verifier in Python with `pySerial`¹ for serial communication and the `pyelftools`² to handle the ELF file. There are two possibilities for the verifier to validate the received information. First, the configuration file format can be used to define constraints and rules that are checked. This allows for straightforward implementation of the verifier. Alternatively, complex checks can be manually implemented into the DMA'N'PLAY framework.

In a configuration file, the developer can provide constraints for these variables. With this configuration file and the ELF file of the attested device, the verifier automatically checks the received data stream. No further manual implementation steps are required. The format of the configuration is simple: Variables are identified by their name in the source code. Furthermore, the developer has to provide the variable type and the constraints to be checked. Listing 1.1 shows an example of such a configuration that describes the data structure of the attested memory. In Listing 1.2, restrictions and logical constraints are defined. DMA'N'PLAY supports the following checks: (1) Static checks that have to be fulfilled, i.e., values that may not change, (2) lists of alternatives i.e., different valid values, (3) arbitrary values, i.e., any value is valid, and (4) ranges, i.e., specify ranges of valid values. This functionality can be flexibly adapted to further use cases and checks.

Alternatively, complex checks can also be implemented. As the verifier is developed in Python it is simple to add further checks or develop more complex rules. Even integration into other systems, e.g., back-end and management systems or web services is possible. In contrast to traditional attestation schemes, no preliminary exploration of possible states is required, easing implementation and modifications to existing systems. In remote attestation schemes that use a precomputed list of valid states as hashes, this list has to be recomputed on every change of the attested system. The usage of a memory-safe language like Python for the verifier prevents a compromise of the verifier due to memory corruption. Note that the input provided by the attested device should be considered untrusted during development.

6.3 DMA'N'PLAY TO-GO

The verifier is an external device that has to be directly connected to the attested device. To compensate for limitations induced by this design, we developed an external device called DMA'N'PLAY TO-GO that can relay attestation data to a

¹ <https://github.com/pyserial/pyserial>.

² <https://github.com/eliben/pyelftools>.

Listing 1.1. Example configuration file for verifier

```

1 layout = cstruct.Struct(
2   "p_settings" / p_settings,
3   "dosage_ml" / cstruct.Int16ul,
4   "bolus_steps_ml" / cstruct.Array(9, cstruct.Float32l),
5   "attestation_header" / cstruct.Array(3, cstruct.Int8ul)
6 )

```

Listing 1.2. Example for valid ranges of variables for verifier

```

1 varmap = {
2   "p_settings:syringe_volume_ml": (DataModel.VarType.STATIC, DataModel.VarStatic(30)),
3   "dosage_ml": (DataModel.VarType.RANGE, DataModel.VarRange(0,6)),
4   "bolus_step_index": (DataModel.VarType.ALTERNATIVES, DataModel.VarAlternatives
5     ([0,1,2,3,4,5,6,7,8])),
6   "attestation_header" : (DataModel.VarType.ANY, DataModel.VarAny(None))
7 }

```

different verifier. DMA'N'PLAY TO-GO is an embedded device with a small form factor, little power consumption, and is available at a low price so that it can be integrated into the attested systems. In particular, we used an ATmega328P 8-Bit microcontroller with 16 MHz clock frequency and 2 kB of SRAM [15]. It consumes 2 to 10 mA at full load depending on frequency [60]. By only connecting the attested device with the receiving pin (RX) on the microcontroller, leaving the transmission pin (TX) of the microcontroller unconnected, a one-way connection is ensured.

In our implementation, Bluetooth Low Energy (LE) was used to transmit the data stream. Bluetooth LE is well-suited for this use case as it has a low power consumption and a reach of up to 100 m. Alternatively, also a microcontroller with Wi-Fi functionality such as the ESP8266 [38] or the ESP32 [36] can be used to send the attestation data to a remote server or cloud service. Alternatively, these microcontrollers can also be used as a verifier, even though they do not have enough computational capabilities to analyze the binary of the attested device. Therefore, the verification logic has to be manually implemented on these microcontrollers.

6.4 Integration Guidelines

In this section, we show the steps required to integrate DMA'N'PLAY into a new application and set up the corresponding verifier.

(1) Identify the memory areas to be attested. When compiling the attested application, all relevant memory areas can be moved into a dedicated memory section. In case no source code is available, also the complete memory can be attested. (2) Integrate an attestation header, i.e., a unique, identifiable string of bits, into the attested memory, e.g., the attested memory section. Note that the attestation header does not need to be placed in a specific position as long as it is inside the attested memory region. In case no source code is available, choose an existing and unique bit string within the attested memory to serve as the attestation header. (3) Configure the DMA controller so that it outputs the

attested memory to the bus used for the attestation, e.g., the serial bus. Take care of the respective bus configuration, e.g., transmission speeds. (4) Define valid states of the attested device and develop a configuration file containing a rule set. When using a binary with debug symbols, the verifier can identify variables by their names. For more information on how to build a configuration file, see Sect. 6.2. (5) Set up the verifier. This includes configuring the bus used for the attestation and providing the binary of the attested device along with the configuration file.

These steps allow integrating DMA'N'PLAY into new as well as legacy devices. Note that no source code of the attested device is required. Changing the configuration or updating the binary requires only a subset of these steps, e.g., replacing the binary, or modifying or updating rules in the configuration file.

7 Evaluation

We implement DMA'N'PLAY into the syringe pump and a drone and attest numerous variables of different types to show the capabilities of the verifier. For each device, we design and execute a typical practical attack to show that DMA'N'PLAY is able to detect the compromise of configuration data. We performed timing measurements to show that DMA'N'PLAY has no timing influence on normal operation.

7.1 Case Study

For the evaluation, we integrated the DMA'N'PLAY attestation into two real-world devices, a syringe pump, and a drone, using our integration guidelines. Both devices perform safety-critical tasks and operate under real-time constraints. We then developed full end-to-end examples and integrated a typical vulnerability in each device. We created a configuration file, defining valid states and ranges for both devices. Upon exploiting the vulnerabilities and applying the respective attack, these were immediately detected. There are different methods to respond to a detected compromise. In the case of DMA'N'PLAY, the verifier could also interact directly with the attested device. For example, the verifier can power off the syringe and alert a doctor, and the drone can be excluded from an autonomous swarm.

Syringe Pump. A syringe pump is a medical device that injects medicine into the body of a patient. We enhanced an open-source syringe [89] with DMA'N'PLAY and implemented it using a Nucleo-F446RE development board that features an ARM Cortex-M4 processor. DMA'N'PLAY is continuously monitoring the devices' configuration so that any illegal operations are detected. To do so, we wrote the corresponding configurations for DMA'N'PLAY. Then, we integrated a common vulnerability into the syringe pump: an insecure configuration interface. This configuration interface allows changing the amount of medicine being injected. The attack can have potentially lethal consequences for the patient if

too much or too little medicine is being injected. Such an attack vector is typical for IoT devices: the most common vulnerabilities are weak, guessable, or hard-coded passwords, as well as insecure network interfaces and services [68].

Drone. We integrated DMA'N'PLAY into a Bitcraze Crazyflie 2.1 drone [19]. Drones feature many safety-critical components that are crucial for correct operation. A malfunction or compromise of one of these components can have severe consequences. For the attack, we use the remote control channel of the drone that is used to send new commands and fly the drone directly. For a remote adversary, this is the primary attack vector. Using this channel, we compromised the device and changed multiple critical values: The configuration of the state estimator determines the flight position and stabilization of the drone. This critical component processes the sensor data from the drone and provides the position and movement of the drone. Changing the parameters of the estimator directly influences the flight behavior of the drone. Moreover, as the control and navigational system of the drone are dependent on this system, these modifications allow controlling the drone without directly interfering with its navigational system. In the case of autonomous drones, this attack will be undetected as the autonomous control system remains unmodified. However, this is only an example of an attack: any flight parameters or configurations could be manipulated, resulting in arbitrary attacker-controlled behavior. It would also be possible to directly control the drone by altering its navigational and way-finding system, or compromising the collision warning system so that the drone will collide and crash.

We integrated the DMA'N'PLAY framework into both devices and configured it to monitor critical components and data. To do so, we defined valid states and ranges for variables in a configuration file. We provided this configuration file to the verifier together with the binary of the software running on the attested device. Then, we monitored the normal operation of the device. Upon exploiting the vulnerabilities and applying the respective attack, these were immediately detected by the verifier by raising an alarm. In practice, there are different methods to respond to a detected compromise, e.g., by raising an alert, or, in autonomous settings with multiple devices, isolating a compromised device. In the case of DMA'N'PLAY, the verifier can also interact directly with the attested device, for example by powering it off in the case of the syringe pump or excluding a drone from a swarm in the case of autonomous drones.

7.2 Real-Time Capabilities

By using DMA and an external verification device, which are both independent of the normal operation of the processor, DMA'N'PLAY can even be applied to hard real-time applications. Such systems have strict responsiveness requirements that limit the integration of new security techniques. The data transmission on the DMA controller does not consume any CPU time, because DMA is a dedicated hardware unit with its own access to the memory bus and peripheral interfaces. In general, the CPU and the DMA controller both act as master devices on the memory bus [78, 79]. Since the DMA and the CPU potentially could compete for the usage of the memory bus, round-robin arbitration mechanisms are

implemented in hardware [78,79]. Memory buses can be optimized for either bandwidth or low latency in sharing. For microcontrollers, such as the ARM Cortex-M family, memory buses are optimized for low latency in sharing [78]. Low sharing latency means that very fast switches occur between memory access tasks. In addition, DMA latencies can be precisely predicted [79]. Since peripheral interfaces are relatively slow compared to the speed of the memory bus, the additional operations on the memory bus are negligible. Due to the low latency sharing (round-robin arbitration on the memory bus) implemented between CPU and DMA, as well as the relatively low speed of peripheral interfaces, there is negligible impact on the operations performed by the processor.

7.3 System Performance View

The attestation mechanism via DMA can potentially compete for memory bandwidth with the CPU. As mentioned in Sect. 7.2, the hardware implements round-robin arbitration: Thus, DMA can only occupy up to 50% of the memory bandwidth, but would not starve the CPU on memory accesses. The overall utilization of the memory bandwidth varies from application to application. However, the memory usage of DMA'N'PLAY will be far below the maximum bandwidth usage of 50% as the transmission speeds of typical peripheral buses such as UART or SPI are much less than the memory bus speed. Figure 5 shows the transmission speed of standard UART and SPI configurations.

A typical UART data rate of 115,200 baud is approximately 11.25 kB/s, and an SPI connection clocked with 40 MHz can transfer up to 5000 kB/s. The total capacity of the memory bus is much higher than those achievable over peripherals such as UART and SPI. The Cortex M4 core used in our case study in Sect. 7.1 features a 32-bit AHB Lite Bus for the memory interface [10]. According to the bus specification [9] the transfer consists of one address cycle and at least one or multiple data cycles. Due to the 32-bit bus width, 4 B (32 bit) can be transferred per cycle.

In contrast, SPI is only capable of a single-bit transfer per clock cycle. The SPI peripheral is usually not clocked at full processor speed, but rather at $\frac{1}{2}$ or $\frac{1}{4}$ of the processor clock. Assuming the drone use case from Sect. 7.1 we used a processor clock of 160 MHz on the Cortex M4, the AHB bus clocked at 160 MHz is capable of transmitting 640MB/s. The 40 MHz SPI connection (5 MB) would only take $\frac{1}{128}$ and UART at 115,200 baud would only take $\frac{1}{56888}$ of the available memory bandwidth. Therefore, the memory impact of DMA'N'PLAY is limited and can be configured by selecting adequate transmission speeds.

7.4 Feasibility of Full Memory Attestation

Full memory attestation is feasible, but the attested memory portion and transfer speed always need to be carefully chosen to avoid the well-known time-of-check/time-of-use problem (TOCTOU, see also Sect. 8). Attesting 128 kB of memory with 40 MHz SPI takes about 25 ms. On embedded flash memory chips, there is typically a huge speed difference between reads, which are fast, and

writes, which are very slow. Since flash memory has to be written sector-wise, to write any data, at least one entire sector needs to be erased first and re-written. Erasing a sector of a common chip requires around 50 ms [35]. Slower flash chips even take up to 100 ms [34]. In practice, this circumstance eases attestation as it effectively mitigates the TOCTOU problem.

7.5 Power Consumption

Especially in embedded systems, which are often battery-operated, power consumption is an important aspect. We measured the power consumption of our two prototype implementations for 7 min and measured the total energy consumption. The drone consumed 50 mWh, the syringe pump 46 mWh, adding up to 10 mAh and 9.2 mAh respectively. In summary, we could not measure an increase in the power consumption of the devices running DMA'n'PLAY compared to the default implementation. Hence, DMA'n'PLAY is suitable for mobile applications and small embedded devices. However, integration of DMA'n'PLAY can slightly increase the power consumption of a system due to the influence on the deep-sleep behavior of the processor [83].

Although the integration of DMA'n'PLAY does not influence the power consumption of the attested device, the verifier also requires power. Depending on the processor frequency DMA'n'PLAY To-Go consumes between 2 and 10 mA at full load [60]. While in absolute numbers this seems low, depending on the attested device, the power consumption can be significant. However, the actual power consumption of the verifier depends on the actual implementation, the microcontroller used, and transmission technologies.

7.6 Summary

This evaluation showed the applicability of DMA'n'PLAY using two real-world examples, a syringe pump, and a drone. In our practical evaluation, we showed how DMA'n'PLAY is able to detect attacks. We showed that DMA'n'PLAY does neither increase power consumption nor influence the runtime behavior of the attested device. This allows a wide usage of DMA'n'PLAY, including devices with real-time constraints.

8 Security Discussion

The novel approach of DMA'n'PLAY to remote attestation has several advantages in practicability compared to traditional remote attestation schemes. But several security aspects have to be considered when using DMA'n'PLAY.

Attack Model. For a successful attack, the adversary has to modify memory content, such as variables, without being detected. The DMA controller, which is independent of the software running on the attested device, sends the memory content to the external verifier in a circular process. The verifier continuously monitors this memory content. As discussed in Sect. 3, the attacker cannot

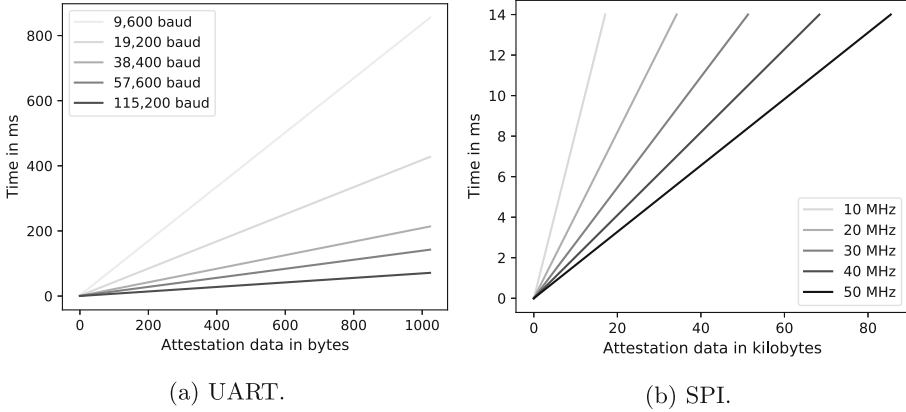


Fig. 5. Trade-off between bus speed, amount of attested data, and transmission time.

influence which memory is being monitored, interrupt this process, or falsify the reported data. Therefore, it is crucial that the adversary cannot change or influence the configuration of the DMA controller. In the following, we discuss possible attacks on attestation and show how DMA’N’PLAY circumvents these.

Time-of-Check/Time-of-Use. The time-of-check/time-of-use problem (TOC-TOU) describes the circumstance that in remote attestation schemes there is a delay between the attestation time, when the integrity of the device is checked, and execution later on. An attacker can exploit this time span to carry out an attack without detection. Also, DMA’N’PLAY suffers from this problem: The memory is copied in a circular process to the verifier, leaving a time span between each time a specific memory location is copied. The length of this interval is dependent on two variables: The amount of data that is being verified and the bus transmission speed. Figure 5 shows this dependency for typical bus speeds of serial interfaces. The faster the transmission speed, the shorter the interval between two attestation runs. Note that the graph does not start at 0B. For DMA’N’PLAY an attestation header is required to identify the memory layout in the data stream. In our implementation, this attestation header takes 3B. While this time span between two attestation runs can be exploited in theory, in practice the attacker does not know which memory parts are currently being attested. As eluded in Sect. 5 the transmission of the memory content to the verifier is performed by the DMA controller which is completely independent of the processor and the software running on it.

Device Authenticity and Offloading. The DMA’N’PLAY device is physically connected to the attested device, therefore the attacker is not able to trick DMA’N’PLAY into attesting a different device (offloading). In order to change the input data to the DMA’N’PLAY device, the attacker has to reconfigure the DMA controller which is not possible according to our Threat model (c.f. Sect. 3).

Limitations. DMA'N'PLAY does only detect deviations from predefined behavior. Therefore, modifications to static code in flash will not be discovered. To ensure integrity of static code, secure boot mechanisms can be used. Secure boot mechanisms are widely used and also available in many microcontrollers.

Presence of Attestation. The attested device and the verifier are connected via a one-way serial connection, i.e., there is no feedback channel from the verifier to the attested device. To change this, a new physical connection between the attested device and the verifier has to be established, a task that has to be performed manually. According to our threat model, physical attacks are out of scope. Therefore, the attested device cannot get any response from the verifier. It makes no difference to the attested device whether a verifier is connected or not. Thus, the attested device cannot check if it is currently being attested or not.

Side-Channels. However, the attested device could use heuristics to determine if it is attested: Due to the necessity for an external verifier device, in many application areas an attacker can use side-channels or heuristics to determine whether it is likely that a verifier is present. For example, in the case of a vehicle, the integrity is probably being checked in a garage during maintenance. A drone is unlikely being attested during flight with a large verifier. Therefore, we developed DMA'N'PLAY TO-GO, a small, embedded device that can be used as an external verifier and be integrated in case a larger external verifier cannot be used. More details on DMA'N'PLAY TO-GO can be found in Sects. 5.3 and 6.3.

Security of DMA-Controllers. The security of DMA'N'PLAY attestation is based on the assumption that the attacker cannot alter the data that the DMA controller sends as eluded in Sect. 5.4. Therefore, we must ensure the integrity configuration of the DMA controller. Protection mechanisms for DMA are different with respect to the targeted platform, and the use case of DMA. In servers and workstations, DMA is used for fast communication across peripherals such as network and graphic cards, usually over PCI(E). Thus, servers and workstations feature specific protections like the input-output memory management unit (IOMMU) [3, 8]. However, such protections are not present on MCUs. MCUs have different architectures and requirements. MCUs are used for embedded applications (e.g., vehicles such as cars and trains, industrial facilities, or IoT deployments) and require lower communication speeds between peripherals than servers or workstations. The DMA controller in embedded contexts unburdens the CPU from wasting scarce CPU time on managing data transfers (e.g., UART or SPI data transmissions). For example, a heavy CPU load can limit the system in its ability to meet scheduled deadlines, important in the embedded context. We extensively elaborate on how to securely lock DMA controllers in Sect. 5.4. We also sum up all requirements and targeted platforms in Sect. 5.5.

Manipulations of the Attestation Header. The attestation header is a crucial component in identifying the components of the attested memory region. This attestation header is controllable by the attacker. This means the attacker can fully manipulate and shift the attestation header. However, the attacker

cannot change the location and amount of the memory being attested as this requires modifying the configuration of the DMA controller. If the attestation header is changed, then it is not recognized by the verifier, resulting in a failed attestation. If the attestation header is moved, this is detected during attestation as the positions in the data stream change. Furthermore, changes to the position of the attestation header cause a misalignment of the attested data, which also causes the attestation to fail.

Attacks on the Verifier. The security of the verifier is crucial for DMA’N’PLAY attestation. Although attacks on the verifier are out of scope, we will briefly discuss security aspects of the verifier. Successful attacks from the attested device to the verifier are unlikely: The attested device and verifier are connected using a one-way serial connection as eluded before. Interrupting this connection or sending modified content will make the attestation fail. The verifier receives a known amount of memory content at a constant rate in a circular process. In this constant process, no complex data structures have to be parsed, and no new memory areas have to be allocated. This makes typical runtime errors highly improbable. The verification process consists of simple comparisons against known information. As the data rate of the serial connection is fixed at a constant rate, denial-of-service (DoS) attacks that jam the verifier are impossible. As explained in Sect. 5 there exists no feedback channel from the DMA’N’PLAY TO-GO to the attested device. Hence, the attacker cannot exfiltrate any information from the verifier, e.g., cryptographic keys or configurations.

Security of DMA’n’Play To-Go. DMA’N’PLAY TO-GO is a low-end embedded device, that receives data from the attested device and relays it to the external verifier. This simple process offers little to no attack surface as the input data is not processed. The fixed transmission rate of the serial connection prevents denial-of-service attacks (DoS) on DMA’N’PLAY TO-GO. If the attacker increases the transmission speed unilaterally, not only will data be incorrectly received by DMA’N’PLAY TO-GO, but the amount of data received will also not increase. Similar to the verifier, there exists no feedback channel from the DMA’N’PLAY TO-GO to the attested device, making the exfiltration of data impossible.

9 Related Work

In the related work, we investigate the security of DMA, as this is a crucial component of DMA’N’PLAY attestation, and give an overview of remote attestation.

9.1 Remote Attestation

In remote attestation, a verifier checks the integrity of a remote, untrusted device. As eluded in Sect. 1 there are different approaches for integrating remote attestation: software-based, hardware-based, and hybrid. Software-based approaches, while being well-suited for legacy devices as they do not have special hardware

requirements, have severe limitations. Their security solely relies on the timing of the responses of the attested device [74]. This induces a complex implementation and strict requirement towards the communication between the prover and the verifier [23]. Moreover, software-based attestation schemes inherently have the problem of a missing root of trust, as the prover does not feature a secret key: So, the verifier cannot identify the attested device, enabling an attacker to relay attestation requests or replace the attested device [23, 74, 81]. Hybrid schemes are popular for embedded and IoT devices, as they address the limitations induced by pure software-based attestation schemes while maintaining less complexity than full hardware-based approaches. Remote attestation can guarantee different security properties. Static attestation schemes ensure the integrity of a device's software and detect manipulations [62]. These attestation schemes cannot detect more sophisticated runtime attacks that do not alter the software, like control-flow [1] or data-flow [2] attacks. In the following, we focus on hybrid attestation schemes.

VRASED is a formally verified attestation framework to perform static attestation [62]. This framework has been extended to cover further security properties. APEX can ensure that specific code has been executed [63]. RATA addresses the well-known time-of-check/time-of-use problem, i.e., the time gap between execution of actual functionality and an attestation run [30]. TinyCFA is a control-flow attestation scheme that is based on VRASED, allowing to attest the correctness of the control flow and detect control-flow attacks [65]. DIALED combines these techniques to perform data-flow attestation and detect data-only attacks [64]. In contrast, LO-FAT performs control-flow attestation completely in hardware, trading higher performance for more hardware complexity [32]. While LO-FAT does not detect non-control-data attacks, the LiteHAX attestation scheme can also cover non-control data-only attacks that do not alter the control flow of the attested application [31]. LiteHAX achieves this by monitoring the execution pipeline of a RISC-V processor.

C-FLAT uses ARM TrustZone to perform control-flow attestation of embedded devices. However, its hash-based approach suffers from the state-space explosion problem and requires dedicated loop handling [1]. OAT also uses TrustZone and introduces the concept of operation execution integrity. In OAT, both control-flow and data-flow of critical parts of a system can be attested, thereby allowing a compromise between security and performance, addressing the problems of efficient verification and state-space explosion [80].

9.2 DMA Security

DMA allows direct access to memory by external devices without involving the main processor, thereby increasing the overall system performance. On the downside, this also facilitates a wide range of attacks, compromising the integrity of a system or reading critical parts of main memory [57]. Well-known examples are attacks via Firewire [18, 33], PCIe [42], and Thunderbolt [72]. This attack vector can, for example, be used to obtain the key of the full disk encryption [20, 33]. DMA controllers can also host malware [76]. Furthermore, external devices

can be untrusted or contain bugs. To counter attacks with external peripheral devices, IOMMU has been introduced [3, 8, 11], enabling memory management for such external devices. However, this has not been proved sufficient [57]. To find vulnerabilities in devices connected via DMA, both, fuzzing [75] and static analysis methods [16] have been proposed.

In addition to direct exploitation, this access can also be used for indirect attacks. Network interfaces are a worthwhile target as they enable remote access. Research found that this enables remote Rowhammer-style attacks [82], which, in contrast to normal Rowhammer attacks, do not need local code execution on the victim [48]. On Intel processors, network cards can even manipulate and observe the processor's last level cache (LLC), allowing remote Prime+Probe [56] attacks to leak critical information [50].

10 Conclusion

In this paper, we present the DMA'N'PLAY framework, that leverages direct memory access (DMA) to directly monitor the memory of the attested device. In contrast to traditional remote attestation schemes, DMA'N'PLAY is capable of directly monitoring the attested device instead of comparing hash values of known benign states. This has multiple advantages: A preliminary investigation of all valid states is not needed, and more complex checks are possible, e.g., bounds checks. Furthermore, the DMA'N'PLAY framework is also suitable for existing legacy devices as neither specialized hardware components nor source code of the attested application is required. We implemented DMA'N'PLAY in two real-world examples, a medical device and a drone, and showed in full end-to-end examples how DMA'N'PLAY can be used to detect compromises of configurations. Furthermore, we provide integration guidelines that explain how DMA'N'PLAY can be integrated into new or existing devices and how to develop configuration files for the verifier to define benign states.

Acknowledgements. This work has been partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—SFB 1119—2366 15297 within project S2. This work was supported by the DFG Priority Program SPP 2253 Nano Security (Project RAINCOAT—Number: 440059533).

References

1. Abera, T., et al.: C-flat: control-flow attestation for embedded systems software. In: 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS). ACM (2016)
2. Abera, T., Bahmani, R., Brasser, F., Ibrahim, A., Sadeghi, A.-R., Schunter, M.: Diat: data integrity attestation for resilient collaboration of autonomous systems. In: 2019 Network and Distributed Systems Security Symposium (NDSS). Internet Society (2019)
3. Abramson, D., et al.: Intel virtualization technology for directed i/o. Intel Technol. J. **10**(3) (2006)

4. Airbus: Operating life. online (2022). <https://www.airbus.com/en/products-services/commercial-aircraft/the-life-cycle-of-an-aircraft/operating-life>
5. Aivaliotis, P., Arkouli, Z., Georgoulas, K., Makris, S.: Degradation curves integration in physics-based models: towards the predictive maintenance of industrial robots. *Robot. Comput. Integr. Manuf.* **71** (2021)
6. Alrawi, O., Lever, C., Antonakakis, M., Monrose, F.: SoK: security evaluation of home-based IoT deployments. In: *IEEE Symposium on Security and Privacy (SP)*. IEEE (2019)
7. Altawy, R., Youssef, A.M.: Security, privacy, and safety aspects of civilian drones: a survey. *ACM Trans. Cyber-Phys. Syst.* **1**(2) (2016)
8. AMD: Amd i/o virtualization technology (iommu) specification. Online (2021). https://www.amd.com/system/files/TechDocs/48882_IOMMU.pdf
9. ARM: Amba 3 ahb-lite protocol specification. Online (2020). https://www.eecs.umich.edu/courses/eecs373/readings/ARM_IHI0033A_AMBA_AHB-Lite.SPEC.pdf
10. ARM: Arm cortex-m4 processor technical reference manual. Online (2020). <https://developer.arm.com/documentation/100166/0001>
11. ARM: Arm system memory management unit architecture specification. Online (2016). <https://documentation-service.arm.com/static/5f900d34f86e16515cdc08fb>
12. ARM: Trustzone technology for armv8-m architecture. Online (2018). <https://developer.arm.com/documentation/100690/latest/>
13. ARM: Configuring and enabling the mmu. Online (2022). <https://developer.arm.com/documentation/den0024/a/The-Memory-Management-Unit/Translating-a-Virtual-Address-to-a-Physical-Address/Configuring-and-enabling-the-MMU>
14. ARM: Trustzone for armv8-a. Online (2019). <https://documentation-service.arm.com/static/602167b6873dd96c4deaf49b>
15. Atmel Corporation: Atmega328p 8-bit avr microcontroller with 32k bytes in-system programmable flash datasheet. Online (2015). https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
16. Bai, J.-J., Li, T., Lu, K., Hu, S.-M.: Static detection of unsafe DMA accesses in device drivers. In: *30th USENIX Security Symposium* (2021)
17. Bartlett, G.: Extending the industrial robot life cycle. Online (2021). <https://www.swri.org/industry/industrial-robotics-automation/blog/extending-the-industrial-robot-life-cycle>
18. Becher, M., Dornseif, M., Klein, C.N.: Firewire: all your memory are belong to us. In: *Proceedings of CanSecWest* (2005)
19. Bitcraze, A.B.: Datasheet crazyflie 2.1 - rev 3. Online (2021). https://www.bitcraze.io/documentation/hardware/crazyflie_2.1/crazyflie_2.1-datasheet.pdf
20. Böck, B., Austria, S.B.: Firewire-based physical security attacks on windows 7, efs and bitlocker. *Secure Business Austria Research Lab* (2009)
21. Brassler, F., Mahjoub, B.E., Sadeghi, A., Wachsmann, C., Koeberl, P.: Tytan: tiny trust anchor for tiny devices. In: *52nd Annual Design Automation Conference*. ACM (2015)
22. Campau, T.: Average age of vehicles in the us increases to 12.2 years, according to s&p global mobility. Online (2022). <https://ihsmarkit.com/research-analysis/average-age-of-vehicles-in-the-us-increases-to-122-years.html>
23. Castelluccia, C., Francillon, A., Perito, D., Soriente, C.: On the difficulty of software-based attestation of embedded devices. In: *2009 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM (2009)

24. Clements, A.A., et al.: Protecting bare-metal embedded systems with privilege overlays. In: IEEE Symposium on Security and Privacy (SP) (2017)
25. Coker, G., et al.: Principles of remote attestation. *Int. J. Inf. Secur.* **10**(2) (2011)
26. Corteggiani, N., Camurati, G., Francillon, A.: Inception: system-wide security testing of real-world embedded systems software. In: 27th USENIX Security Symposium (2018)
27. Costin, A., Zaddach, J., Francillon, A., Balzarotti, D.: A large-scale analysis of the security of embedded firmwares. In: 23rd USENIX Security Symposium (2014)
28. Das, S., Zhang, W., Liu, Y.: A fine-grained control flow integrity approach against runtime memory attacks for embedded systems. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **24**(11) (2016)
29. Dawoud, D.S., Dawoud, P.: *Serial Communication Protocols and Standards RS232/485, UART/USART, SPI, USB, INSTEON*. River Publishers, Wi-Fi and WiMAX (2020)
30. De Oliveira Nunes, I., Jakkamsetti, S., Rattanavipanon, N., Tsudik, G.: On the toctou problem in remote attestation. In: 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS). ACM (2021)
31. Dessouky, G., Abera, T., Ibrahim, A., Sadeghi, A.-R.: Litehax: lightweight hardware-assisted attestation of program execution. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE (2018)
32. Dessouky, G., et al.: Lo-fat: low-overhead control flow attestation in hardware. In: 54th Annual Design Automation Conference (DAC). ACM (2017)
33. Dornseif, M.: Owned by an ipod: Firewire/1394 issues. In: CanSecWest Security Conference CORE05 (2005)
34. elm-tech: Gd25q32 datasheet. Online (2014). <https://datasheetspdf.com/pdf-file/861582/ELM/GD25Q32/1>
35. elm-tech: Gd25q32c datasheet. Online (2020). <http://www.elm-tech.com/en/products/spi-flash-memory/gd25q32/gd25q32.pdf>
36. Espressif Systems: Esp32 technical reference manual. Online (2020). https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
37. Espressif Systems: Esp32-c3 technical reference manual. Online (2022). https://www.espressif.com/sites/default/files/documentation/esp32-c3_technical_reference_manual_en.pdf
38. Espressif Systems: Esp8266 technical reference manual. Online (2020). https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf
39. Falliere, N., Murchu, L.O., Chien, E.: W32. stuxnet dossier. White paper, symantec corp., security response, vol. 5, no. 6 (2011)
40. Farwell, J.P., Rohozinski, R.: Stuxnet and the future of cyber war. *Survival* **53**(1) (2011)
41. Feng, B., Mera, A., Lu, L.: P2IM: scalable and hardware-independent firmware testing via automatic peripheral interface modeling. In: 29th USENIX Security Symposium. USENIX Association (2020)
42. Frisk, U.: Direct memory attack the kernel. In: Proceedings of DEFCON, vol. 24 (2016)
43. Gemalto: The state of IoT security. Online (2018). <https://www.infopoint-security.de/media/gemalto-state-of-iot-security-report.pdf>
44. GNU Project - GNU Compiler Collection: Specifying attributes of variables. Online (2022). <https://gcc.gnu.org/onlinedocs/gcc-11.3.0/gcc/Variable-Attributes.html#Variable-Attributes>

45. Infineon: How to use direct memory access (DMA) controller in traveo ii family. Online (2021). [https://www.infineon.com/dgdl/Infineon-AN220191_How_to_Use_Direct_Memory_Access_\(DMA\)_Controller_in_Traveo_II_Family-ApplicationNotes-v07_00-EN.pdf](https://www.infineon.com/dgdl/Infineon-AN220191_How_to_Use_Direct_Memory_Access_(DMA)_Controller_in_Traveo_II_Family-ApplicationNotes-v07_00-EN.pdf)
46. Infineon: Mpu_memory_protection for kit_aurix_tc297_tft. Online (2020). <https://www.infineon.com/dgdl/?fileId=5546d46274cf54d50174da37dc1d222e>
47. Infineon: Mpu_memory_protection for kit_aurix_tc297_tft. Online (2017). <https://www.nxp.com/docs/en/supporting-information/BL-Micro-NXP-Microcontroller-Overview-James-Huang.pdf>
48. Kim, Y., et al.: Flipping bits in memory without accessing them: an experimental study of dram disturbance errors. *ACM SIGARCH Comput. Archit. News* **42**(3) (2014)
49. Koscher, K., et al.: Experimental security analysis of a modern automobile. In: *IEEE Symposium on Security and Privacy (SP)*. IEEE (2010)
50. Kurth, M., Gras, B., Andriess, D., Giuffrida, C., Bos, H., Razavi, K.: Netcat: practical cache attacks from the network. In: *IEEE Symposium on Security and Privacy (SP)*. IEEE (2020)
51. Kwon, D., Shin, J., Kim, G., Lee, B., Cho, Y., Paek, Y.: uxom: Efficient execute-only memory on arm cortex-m. In: *28th USENIX Security Symposium*. USENIX Association (2019)
52. Langner, R.: Stuxnet: dissecting a cyberwarfare weapon. *IEEE Secur. Privacy* **9**(3) (2011)
53. Lee, D., Kohlbrenner, D., Shinde, S., Asanović, K., Song, D.: Keystone: an open framework for architecting trusted execution environments. In: *15th European Conference on Computer Systems (EuroSys '20)*. ACM (2020)
54. Leens, F.: An introduction to I2C and SPI protocols. *IEEE Instrum. Meas. Mag.* **12**(1) (2009)
55. Levy, A., et al.: Multiprogramming a 64kb computer safely and efficiently. In: *26th Symposium on Operating Systems Principles, SOSP '17*. ACM (2017)
56. Liu, F., Yarom, Y., Ge, Q., Heiser, G., Lee, R.B.: Last-level cache side-channel attacks are practical. In: *IEEE Symposium on Security and Privacy (SP)*. IEEE (2015)
57. Marketos, T., et al.: Thunderclap: exploring vulnerabilities in operating system IOMMU protection via DMA from untrustworthy peripherals (2019)
58. Mera, A., Feng, B., Lu, L., Kirda, E.: Dice: automatic emulation of DMA input channels for dynamic firmware analysis. In: *IEEE Symposium on Security and Privacy (SP)*. IEEE (2021)
59. Mera, A., Chen, Y.H., Sun, R., Kirda, E., Lu, L.: D-box: DMA-enabled compartmentalization for embedded applications. In: *2022 Network and Distributed Systems Security Symposium (NDSS)*. Internet Society (2022)
60. Microchip Technology Inc: Atmega48a/pa/88a/pa/168a/pa/328/p. Online (2018). <https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf>
61. Motorola Inc: SPI block guide v03.06. Document number S12SPIV3/D (2003)
62. Nunes, I.D.O., Eldefrawy, K., Rattanavipanon, N., Steiner, M., Tsudik, G.: Vrsed: a verified hardware/software co-design for remote attestation. In: *28th USENIX Security Symposium* (2019)
63. Nunes, I.D.O., Eldefrawy, K., Rattanavipanon, N., Tsudik, G.: Apex: a verified architecture for proofs of execution on remote devices under full software compromise. In: *29th USENIX Security Symposium* (2020)

64. Nunes, I.D.O., Jakkamsetti, S., Tsudik, G.: Dialed: data integrity attestation for low-end embedded devices. In: 58th ACM/IEEE Design Automation Conference (DAC). IEEE (2021)
65. Nunes, I.D.O., Jakkamsetti, S., Tsudik, G.: Tiny-CFA: minimalistic control-flow attestation using verified proofs of execution. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE (2021)
66. NXP: Examples of setting the DMA controller on the power architecture mpc5675k family of microcontrollers. Online (2012). https://www.nxp.com/docs/en/application_note/AN4522.pdf
67. Osborne, A.: *Introductions to Microcomputers*; vol. 1. Basic Concepts, McGraw-Hill Osborne Media (1980)
68. OWASP: Internet of things (IoT) top 10 2018 (2018). <https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf>
69. Quarta, D., Pogliani, M., Polino, M., Maggi, F., Zanchettin, A.M., Zanero, S.: An experimental security analysis of an industrial robot controller. In: IEEE Symposium on Security and Privacy (SP). IEEE (2017)
70. Reilly, E.D.: *Memory-Mapped I/O*. Wiley, Hoboken (2003). ISBN 0470864125
71. RISC-V: The RISC-V instruction set manual volume ii: privileged architecture. Online (2017). <https://riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf>
72. Ruytenberg, B.: Breaking thunderbolt protocol security: vulnerability report. Online (2020). <https://thunderspy.io/assets/reports/breaking-thunderbolt-security-bjorn-ruytenberg-20200417.pdf>
73. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: what it is, and what it is not. In: 2015 IEEE Trustcom/BigDataSE/ISPA, vol. 1. IEEE (2015)
74. Seshadri, A., Perrig, A., van Doorn, L., Khosla, P.K.: Swatt: software-based attestation for embedded devices. In: IEEE Symposium on Security and Privacy (SP). IEEE (2004)
75. Song, D., et al.: Periscope: an effective probing and fuzzing framework for the hardware-OS boundary. In: 2019 Network and Distributed Systems Security Symposium (NDSS). Internet Society (2019)
76. Stewin, P., Bystrov, I.: Understanding DMA malware. In: Flegel, U., Markatos, E., Robertson, W. (eds.) DIMVA 2012. LNCS, vol. 7591, pp. 21–41. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37300-8_2
77. STMicroelectronics: Managing memory protection unit in stm32 mcus. Online (2021). https://www.st.com/resource/en/application_note/dm00272912-managing-memory-protection-unit-in-stm32-mcus-stmicroelectronics.pdf
78. STMicroelectronics: Using the stm32f0/f1/f3/gx/lx series DMA controller. Online (2020). https://www.st.com/resource/en/application_note/cd00160362-using-the-stm32f0f1f3gxlx-series-dma-controller-stmicroelectronics.pdf
79. STMicroelectronics: Using the stm32f2, stm32f4 and stm32f7 series DMA controller. Online (2016). https://www.st.com/resource/en/application_note/dm00046011-using-the-stm32f2-stm32f4-and-stm32f7-series-dma-controller-stmicroelectronics.pdf
80. Sun, Z., Feng, B., Lu, L., Jha, S.: Oat: attesting operation integrity of embedded devices. In: IEEE Symposium on Security and Privacy (SP). IEEE (2020)
81. Surminski, S., Niesler, C., Brassler, F., Davi, L., Sadeghi, A.-R.: Realswatt: remote software-based attestation for embedded devices under realtime constraints. In: 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS). ACM (2021)

82. Tatar, A., Konoth, R.K., Athanasopoulos, E., Giuffrida, C., Bos, H., Razavi, K.: Throwhammer: rowhammer attacks over the network and defenses. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18) (2018)
83. Texas Instruments Incorporated: Direct memory access (DMA) controller module. Online (2018). <https://www.ti.com/lit/ug/slau395f/slau395f.pdf>
84. The LLVM Compiler Infrastructure Project: Attributes in clang. Online (2022). <https://clang.llvm.org/docs/AttributeReference.html#variable-attributes>
85. Valmari, A.: The state explosion problem. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 429–528. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_21
86. Van der Veen, V., et al.: Practical context-sensitive CFI. In: 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS). ACM (2015)
87. Wenzl, M., Merzdovnik, G., Ullrich, J., Weippl, E.: From hack to elaborate technique—a survey on binary rewriting. *ACM Comput. Surv. (CSUR)* **52**(3) (2019)
88. Wetzels, J.: The RTOS exploit mitigation blues. Online (2017). <https://hardware.io/document/rtos-exploit-mitigation-blues-hardware-io.pdf>
89. Wijnen, B., Hunt, E.J., Anzalone, G.C., Pearce, J.M.: Open-source syringe pump library. *PLoS ONE* **9**(9) (2014)



Recommendation for a Holistic Secure Embedded ISA Extension

Florian Stolz¹✉ , Marc Fyrbiak^{2,3} , Pascal Sasdrich¹ ,
and Tim Güneysu¹ 

¹ Ruhr-Universität Bochum, Bochum, Germany

{florian.stolz,pascal.sasdrich,tim.guneysu}@rub.de

² Max Planck Institute for Security and Privacy, Bochum, Germany

marc.fyrbiak@mpi-sp.org

³ emproof GmbH, Bochum, Germany

Abstract. Embedded systems are a cornerstone of the ongoing digitization of our society, ranging from expanding markets around IoT and smart-X devices over to sensors in autonomous driving, medical equipment or critical infrastructures. Since a vast amount of embedded systems are safety-critical (e.g., due to their operation site), security is a necessity for their operation. However, unlike mobile, desktop, and server systems, where adversaries typically only act have remote access, embedded systems typically face attackers with physical access. Thus embedded system require an additional set of defense techniques, preferably leveraging hardware acceleration to minimize the impact on their stringent operation constraints. Over the last decade numerous defenses have been explored, however, they have often been analyzed in isolation. In this work, we first systematically analyze the state of the art in defenses for both software exploitation and fault attacks on embedded systems. We then carefully design a holistic instruction set extension to augment the RISC-V instruction set architecture with instructions to deter against the threats analyzed in this work. Moreover we implement our design using the *gem5* simulator system and a binary translation approach to arm software with our instruction set extension. Finally, we evaluate performance overhead on the *MiBench2* benchmark suite. Our evaluation demonstrates a ROM overhead increase of 20% to defeat the aforementioned attacks.

Keywords: Embedded Security · Physical Attacks · ISA Extension

1 Introduction

With the ubiquitously expanding Internet of Things (IoT), the demand for embedded devices continuously increases. However, such an ubiquitous presence of embedded systems in also security-relevant appliances inevitably increases

the potential for attacks. Through physical access, adversaries can particularly attack digital devices and security-critical systems through implementation attacks, such as Side-Channel Analysis (SCA), Fault Injection Attack (FIA). Along with (limited) software-induced attacks, the range of threats that modern embedded devices face is broad and multifaceted. Physical access allows adversaries to perform glitching attacks, which may lead to bit flips in the fetched instructions. These faulty bits may cause erroneous instructions to be executed or may even change the semantics to a No-Operation (NOP), effectively skipping an instruction [2]. Especially in the context of cryptography, such glitches can have severe consequences such as key leakage via Differential Fault Attack (DFA) [24]. Furthermore, glitch-induced NOPs by physical adversary may violate the control-flow of the program, which in turn can leak secrets. On the software side, adversaries can mount software-based attacks to manipulate the control flow of the program, for example, by overwriting the return address. Over the last decades, many countermeasures [3, 11, 15, 22, 33, 34] to these threats have been developed, but have mostly been studied in isolation. However, as seen above, in the case of embedded systems, the adversary can mount an attack using many different techniques. Therefore, an embedded system has to employ a combination of defenses to deter these adversaries. Simply stacking different approaches on top of each other may give rise to inefficiencies, as they may reimplement similar primitives instead of sharing a common base. Instead it is preferable to determine an efficient set of instructions which can achieve a maximum of security by reusing primitives.

Goal and Contributions. In this work, we focus on an embedded system defense to protect against both software-based exploitation and glitching attacks simultaneously. Our goal is to design an instruction set extension with a particular focus on RISC-V to minimize performance impacts. To this end, we first systematically analyze the state of the art for both aforementioned attack strategies. Based on elaborated insights, we then carefully design our instruction set extension to leverage synergies between different defenses, i.e. employing a glitching defense to facilitate higher-level defenses. By hashing the current instruction stream, we create a label-based Control Flow Integrity (CFI) scheme to protect forward-edges as well as a pointer protection scheme to defend backward-edges against Return-Oriented Programming (ROP) attacks. Finally, we implement and evaluate our instruction set extension using binary translation and demonstrate an average memory overhead of 20% and average performance overhead of 28%. In summary our contributions are:

- **Systematic Analysis.** We carefully analyze state-of-the-art hardware-accelerated defenses against instruction glitching attacks and memory corruption vulnerabilities. Based on our analysis, we then work out defense combinations to leverage their advantages to maximize security guarantees, while minimizing potential performance impact.
- **Novel Hardware Extension.** Based on our systematic analysis, we design a novel instruction set extension for RISC-V that defeats the aforementioned attacks and induces minimal overhead. In particular, our extension leverages

an anti-glitching defense that ensures basic block instruction stream integrity. Each basic block hash is then used to implement a label-based CFI defense to protect forward control-flow edges. Furthermore, we use a pointer protection to secure backward control-flow edges.

- **Evaluation.** We evaluate our recommendation using the *gem5* simulator and the *MiBench2* benchmark suite. We then compare our solution to existing works, which aim to secure embedded systems against similar threats, and find that our solution has a 39% lower memory overhead while having a 81% higher performance overhead.

2 Technical Background

In the following, we provide a concise background on most prominent attack vectors for embedded systems, namely (1) software-based attacks via code injection and re-use, and (2) fault attacks via glitching.

2.1 Code Injection & Reuse Attacks

Even though processing technology made significant progress, embedded systems are still typically constrained in both resources and features. Moreover, C and C++ are still the predominant languages and they do not provide any memory safety features and thus improper use of memory allocations leads to catastrophic exploits (e.g., buffer overflows on both the stack and heap can be used to mount code injection attacks). Advances in the last 20 years, such as DEP and Stack Canaries, made this type of attack more challenging to perform. Nowadays, these countermeasures can also be found in embedded CPUs, which usually offer basic memory protection in form of non-executable memory regions. This restricts attackers to leverage so-called code reuse attacks, f.i., ROP-based exploitation [9].

2.2 Glitching Attacks

Adversaries are especially powerful when granted physical access to the target device as this enables to challenge various assumptions (e.g., the integrity of the instruction stream). By performing a *fault attack*, for example, via *glitching* the clock source or via electromagnetic pulse, an adversary can disturb instruction stream integrity. Bitflips caused by glitches manipulate the data embedded into instructions or change instruction semantics entirely. For example, under certain conditions an instruction can be changed to an NOP instruction. Note that this has severe impacts on the control flow of the program if the skipped instruction is a branch or a comparison [26]. In cryptographic algorithms, such glitches can induce exploitable weaknesses to break all security guarantees. For example, flipping bits during the key addition step may allow attackers to perform a DFA. The possibility of glitching attacks and their effects on program

execution have been studied extensively times in literature. Most recently, Spensky *et al.* [26] analyzed physical attacks under simulated and real conditions. For already existing hardware architectures, special programming techniques are used to prevent such vulnerabilities. Defensive programming techniques were previously analyzed by Wittemann *et al.* [30]. Furthermore, compiler modifications such as presented by Barry *et al.* [3] deter glitching, for example, via instruction duplication. Spensky *et al.* [26] combined different methods to defend against glitches as a *LLVM* extension, including the previously mentioned techniques of code redundancy. A major problem of these approaches is the induced overhead both in the code size and time domain.

3 Preliminaries

We now introduce the fundamentals for the scope of our work, including the assumed system model and adversary model. Based on the models, we then define the security goals that should be achieved by a holistic Instruction Set Architecture (ISA) extension in the context of embedded systems.

3.1 System Model

As outlined before, we focus on a common embedded system model (e.g., typical for many IoT applications): a resource-constrained System-on-a-Chip (SoC) including a processor and other important peripherals such as Random-access memory (RAM) and Read-only memory (ROM), f.i. with memory in the sub one megabyte range and processing speeds of up to 200 MHz. Moreover we assume a system with an Memory Protection Units (MPUs) to offer basic memory protection. Examples of such systems are the Cortex-M range by *ARM* and many RISC-V based products. Throughout this work, we assume that the software runs in a *bare-metal* fashion on the embedded systems, i.e. no real-time operating system is available for the sake of simplicity. However, we want to emphasize that the with a context switch, the real-time operating system support can be added as well.

3.2 Adversary Model

We assume an attacker with physical access to the target system, i.e. to analyze the Printed Circuit Board (PCB) and peripherals. Moreover, we assume that the attacker is able to mount physical attacks by means of fault injection via glitching. However, attacks on the microarchitecture itself and fault injection via laser are out-of-scope of our work. Consequently, we assume the integrity of the integrated RAM/ROM. Exploitation is thus only possible by manipulating signals or sending malicious inputs to the device, i.e. to leverage a software bug for exploitation. The high-level goal of the adversary is to exploit a given device because of private or economic incentive (e.g., forcefully unlock (unintended) features).

3.3 Security Goals

The nature of our adversary model implies protection against various attacks vectors that can be leveraged for exploitation.

Anti-Glitching. Firstly, the integrity of executing instructions should be guaranteed. Otherwise, an adversary can introduce disturbances via glitching to affect the execution. Note that in the context of embedded systems, anti-glitching techniques should be tightly coupled to other defenses, such as CFI, as glitches itself are able to lead to an invalid control flow.

Control Flow Integrity. Secondly (arbitrary) attacker-controlled code execution should be prevented. Note that a properly configured Physical Memory Protection(PMP) on RISC-V processors can already stop basic code injection attacks by disallowing execution in certain regions, such as the stack. However, code-reuse attacks, such as ROP, are still possible. Thus, CFI is a vital requirement for system security. Especially on resource constrained devices, hardware-accelerated CFI on the ISA level may be viable, as code instrumentation creates significant performance impact.

Memory Safety. Lastly, memory corruption should be prevented. Memory unsafe languages such as *C* facilitate exploitation of software bugs (e.g., to overflow memory buffers or write to unwanted memory locations). Note this leads to control-flow changes that may not be detectable by a CFI scheme. Under our adversary model, memory safety has to be combined with other defenses, since perfect memory safety does not eliminate CFI violations as instruction-skips may also be used to mount attacks.

4 Literature Study

We now provide a concise summary of the state of the art of various defenses that have been proposed over the years. Based on our summary, we then discuss which defenses achieve best synergy effects in our system and adversary model.

4.1 Glitching Defenses

The increasing reliance on embedded systems in, for example, IoT appliances or critical infrastructure also prompts for increased security measures in these devices. Considering the potentially severe consequences of glitching attacks as mentioned in Sect. 2.2, several hardware accelerated countermeasures have been developed. In the context of embedded systems, defenses against fault attacks are often combined with other techniques to facilitate CFI. This stems from the fact that skipping control flow instructions can cause control flow violations. In this section, we exclude CFI and focus on instruction stream integrity. Note that we discuss how these primitives may facilitate CFI in detail in Sect. 4.2.

In general, we divide glitching attack defenses into three groups based on their underlying mechanism: anomaly detection, instruction chaining and instruction hashing. As noted before, we focus especially on instruction skips caused by glitching.

Anomaly detection. This defense techniques aims to alert the processor about a possible glitch attack by observing its environment and scanning it for potential indicators of an attack, such as voltage drops. Yuce *et al.* [33] demonstrated *FAME*, a co-processor which triggers a software handler upon detecting a fault. The co-processor is composed of a fault detection unit, which monitors several aspects of the Central Processing Unit (CPU), such as the clock signal. After detecting a fault, *FAME* stops the execution of the main processor. It enters a *safe mode*, where a trap handler can recover the fault. A major advantage of this approach is its low footprint in code size and execution time. The original program does not have to be recompiled and only a fault handler has to be added.

Instruction chaining. Bitflips manifest themselves in a changed instruction word, for example, changing a branch to a NOP. These changes usually only have a limited impact on the surrounding instructions, as each instruction is essentially executed in isolation. Therefore, making instructions depend on each other can spread the effects of glitching attacks out, which usually leads to unrecoverable corruption of the instruction stream and thus deter a possible attack. This approach is used by Werner *et al.* [27] to protect processors from physical attacks. They describe a mechanism based on authenticated encryption, which accumulates a state based on the execution history and is used to decrypt each instruction as it enters the pipeline. A fault will be detected when an invalid instruction is decoded. Alternatively, an integrity verification may be performed to stop randomly decrypted instructions from executing. Similarly the works by Savry *et al.* [22] and de Clercq *et al.* [8] use a mask to decrypt instructions. The first approach uses a static mask, which changes based on a permutation during the execution, the latter work uses a mask based on several values such as the previous program counter location. Notably, the masks employed by Savry *et al.* are not depending on the execution history, which may allow for malicious modifications using dedicated managing instructions. However, the integrity of the initial mask is secured via a Message Authentication Code (MAC).

Instruction hashing. Hashes are commonly used to check the integrity of data. Thus, hashing instructions is a straightforward method to monitor the integrity of the instruction stream. Fei *et al.* [13] employ a standalone hardware monitor, which accumulates a hash during the execution of a single basic block. At the end of each block, the hash is compared to an internal table storing the expected hash. In case of a mismatch, an exception is raised. The authors mention the use of *MD5* or *SHA-1*, however, their demonstration uses a simple *XOR* hash. Rodriguez *et al.* [21] developed an architecture, which stores the expected hash

and further attributes in the instruction stream. Similarly to the previous solution, the computed hash is compared to the expected hash. The block length is encoded into the attributes to prevent attackers from skipping the hash comparison. The authors suggest a *XOR* or Linear-Feedback Shift Register (LFSR) as the hash function. A similar approach was used by Ohlsson *et al.* [19], who use a Cyclic Redundancy Check (CRC) as the hashing function. Werner *et al.* [28] systematically evaluated the requirements for an appropriate hash function and found that a CRC is best suited for instruction hashing. A major problem of these approaches compared to the other fault defenses, is their latency. As the hash is only checked at the end of a block, the fault goes unnoticed for the remainder of the basic block. This can be accounted for by embedding a small hash into each instruction, which is checked during the execution of each instruction, as proposed by Wilken *et al.* [29]. However, this approach either requires a complete ISA redesign or some kind of hardware unit with access to each hash. The latter was implemented by Werner *et al.* [28] as a memory-mapped peripheral on a *ARM* processor.

Discussion. Compared to software-only solutions, ISA and hardware-level can offer considerable overhead reductions. Nevertheless, not all of the previously mentioned techniques may be combined intuitively with other solutions or come with other downsides. Standalone hardware monitors such as *FAME* [33] or the proposal by Fei *et al.* [13] have the advantage that they do not require modification to the ISA itself. However, these monitors have to constantly detect basic blocks, for example, by scanning for branch instructions. It would thus be preferable to give the processor built-in capabilities to differentiate between basic blocks. Furthermore, internal storage required by these solutions is limited and needs managing and data swapping, as described by Fei *et al.* In contrast to this, integrated instruction chaining or hashing avoid these problems and come with the advantage of being building blocks for other structures. Many of the previously proposed schemes are used to implement protection mechanisms such as CFI. They are thus ideal candidates for a holistic security extension. The main differences between chaining and hashing lie in their latency. Compared to hashing, chaining inhibits low latency as a bitflip will immediately result in randomly decrypted instructions. Yet this behavior may not always be desirable, especially if the processor may be in an elevated execution state. Therefore, further modifications are required to prevent random instruction execution. Furthermore, chaining is often based on encryption, which by itself usually results in major performance degradation or hardware overhead. Instruction hashing avoids these specific problems by leaving the individual instructions intact and performing the hashing operating in parallel. Additionally, the hashing function may be more compact in hardware than an encryption scheme. The latency problems of instruction hashing may be solved by using continuous hashing at the cost of a major ISA redesign.

4.2 CFI

Not only physical attacks pose a serious threat to embedded devices, but also more conventional software-based attacks. Whereas straight-forward code injection attacks are stopped by using technologies such as the MPU, more complex attacks like Code Reuse Attack (CRA) require additional defenses. CFI has been thoroughly explored as a countermeasure against such CRAs. This technique aims to enforce the Control Flow Graph (CFG) of a program by checking the forward edges, created by jumps, and the backward edges, caused by returns. A *fine-grained* approach strictly enforces the CFG, but usually creates a large performance overhead, whereas *coarse-grained* CFI relaxes the rules to gain performance at the cost of security [9].

We divide existing hardware accelerated solutions into three categories: State-based, Policy-based and Heuristics-based. However, in practice many solutions combine techniques from two or all categories.

Policy-based. During its normal execution, a program usually follows some predictable patterns. For example, by definition, a branch targets the entry of a basic block. This rule is broken by CRAs, which execute small code snippets within basic blocks. Thus, enforcing this rule thwarts some control-flow attacks and is used in many solutions like Intel CET [25], which uses the ENDBRANCH instruction to terminate an indirect branch. Alternative solutions prohibit arbitrary branches between functions, for example, via a label, which is placed at each call/return site and is checked during each control-flow transfer. If the expected label does not match, a control-flow violation occurs. This approach is used by Christoulakis *et al.* [7] who employ the branch-delay slot to efficiently load the expected label during a branch. The label can also be generated implicitly using the *Instruction hashing* explained in Sect. 4.1 based on the execution history.

State-based. If an attack follows the rules outlined above, it will stay undetected degrading the security of the system. This may happen if two functions have the same label, which allows for two legit backward edges. To overcome this problem many CFI solutions include a state, which is checked on a regular basis. The most popular solution is a Shadow Call Stack (SCS) [6]. Every call pushes the return address to the regular stack as well as to the SCS. Upon a return, the addresses on the stack and SCS are compared. If they do not match, an exception is thrown. The SCS may be implemented as a hardware stack or as a second stack located in RAM. Davi *et al.* [11] introduced an alternative approach by forcing the function to return to an active call site. Each function is assigned a label, which is activated by an instruction at the start of the function. When returning from a callee it is checked if the label is still active. If not, an adversary changed the return address to a function which was not previously active. Alternatively, the *instruction chaining* approaches from Sect. 4.1 can be used to implement CFI. Encrypting instructions and making them dependent on the previous instructions stops the attacker from arbitrary executing existing code.

Heuristics-based. Lastly, hardware monitors can be employed, which screen the execution history for unusual behaviour, such as short instruction sequences followed by returns indicating a ROP attack. A major problem however is the correct selection of the underlying heuristic. Solutions such as by Kayaalp *et al.* [15] use multiple thresholds to allow for variable gadget length, which lowers the false positive rate.

Discussion. A good CFI solution should provide some form of forward as well as backward-edge protection. Heuristic-based solutions cannot provide such a protection as they detect attacks based on the execution characteristics. Furthermore, it has been shown that monitors which aim to detect short instruction sequences between branches, which are typical for short gadgets, can be circumvented. Thus, heuristics do not seem to be a viable candidate for general-purpose microcontrollers. Most CFI solutions use a mixture of different techniques to achieve both backward- and forward-edge protection. The most promising candidates for backward-edge protection are SCSs as well as HAFIX [11]. However, both require special attention to support common programming paradigms such as recursion and, in the case of SCSs, exceptions. Ideally, SCS data should be placed in processor internal memory, which however is limited, in term requiring some kind of on-demand loading. Therefore, to minimize overhead, a label-based policy seems viable for embedded systems. This also allows us to reuse *instruction hashing*. Notably, this only protects forward edges. Backward-edge violations occur when overwriting the return address, thus requiring a memory protection as discussed below.

4.3 Memory Integrity

In case of a remote attacker in a IoT scenario, a memory vulnerability, such as a buffer overflow, may be used to overwrite data like return addresses on which the control-flow depends. In this paper we refer defenses which guard against such memory attacks as *memory integrity*. As before, we divide existing work into three categories: detection, pointer protection and tagging.

Detection. Buffer overflows actively change the data surrounding the original buffer. Many solutions aim to detect such overflows by placing control values in front of important data like the return address. These are commonly referred to as *stack canaries*. Before each return, the control value is compared to its expected value. If the values do not match an exception is raised [10].

De *et al.* [12] developed a RISC-V extension, which employs a Physically Unclonable Function (PUF) and a binary secret to generate canaries which are placed at each buffer. Thus, their detection method can detect buffer overflows even if they do not overwrite the return address, which is usually the only the checked location.

Pointer Protection. Alternatively, pointers may be protected so that they cannot be simply overwritten by the attacker. In 2016 ARM introduced *Pointer Authentication* as an ISA extension to ARMv8.3-A [23]. It is based on the realization that the upper bytes of a 64-bit pointer are essentially unused and can store additional information. They place a short MAC, also referred to as Pointer Authentication Code (PAC) into the upper bits of the pointer, which is computed using the current context, such as the stack pointer, and domain specific keys. Before using the pointer, an authentication instruction verifies the PAC and makes the pointer invalid if the process fails. On the contrary, *Pointer Encryption* modifies the whole pointer by encrypting it before placing it into memory. The approach by Zhu *et al.* [34], uses a *GCC* extension which automatically protects relevant pointers by applying a XOR-encryption with a key derived from the memory location and a dynamic runtime key. A downside of this approach is the random execution which is inevitably caused by any kind of corruption or manipulation of the encrypted pointer, as the plaintext pointer will be random. Lastly, *Pointer Capabilities* add specific access rights and constraints to each pointer, so that out of bounds writes become impossible. The CHERI ISA [32] represents a major project which uses a 256-bit pointer format to encode various information about each pointer, such as the base address, length and permissions.

Isolation. In the following, we briefly describe approaches which isolate memory regions from each other, so that they cannot interfere. Most microcontrollers already offer basic functionality for memory isolation through a MPU on ARM or the PMP on RISC-V. However, the amount of memory regions as well as available access (e.g., user-mode and supervisor-mode) are limited. Furthermore, even in the presence of unique process identifiers, process internal memory isolation remains a problem. The ARM MTE extensions [1] introduced in ARMv8 provide memory tagging by assigning a 4-bit tag to every 16 bytes in memory. Additionally, each 64-bit pointer stores a corresponding tag in its most significant byte. Only if the tags match, a read or write are possible. This can not only protect against buffer overflows, but also against use-after-free attacks. The approach by Bradbury *et al.* [5], implements tagging for RISC-V and uses 2-bit tags for every 8 bytes in memory. Unlike ARM's solution, the tags do not correspond to a key, but to access permissions such as read- or write-only. Kim *et al.* [16] presented RIMI, which extends upon RISC-V's PMP and add instructions which only allow control-transfers and load/stores inside one of two domains. All relevant instructions are duplicated for *domain0* and *domain1*. This approach is especially viable on embedded systems with limited resources, as no additional tag bits are required. Instead, the isolation is achieved through specialized instructions as well as existing and added hardware units.

Discussion. Many memory protection schemes are built with larger systems in mind, which becomes a problem in the context of resource-constrained embedded systems. Capability-based systems can offer a fine-grained protection at the cost

of increasing the necessary space required for a single pointer and increased complexity. Even compressed capability formats such as CHERI Concentrate [31] take up 64 bits for 32-bit architectures. A similar problem applies to ARM PAC as well as ARM MTE as they abuse the fact, that the upper bytes of 64-bit pointers are typically unused. On embedded 32-bit platforms placing a sufficiently secure MACs becomes impossible. Most of the time, the whole 32-bit space is used with only a few bits being available. Isolation through mechanisms such as RIMI are suitable for embedded systems, but require a general rewrite of the software. Memory Tagging in general seems to be a good fit for embedded systems but comes with some downsides such as limited tag space. Naturally, a trade-off must be made between available tag bits, which allows for more fine-grained control, and the acceptable memory overhead. For example, ARM's solution using 4 bits for every 16 bytes necessarily occupies 8 kB out of a 256 kB RAM module. Furthermore, a software or hardware component is required to manage the tags and schedule them accordingly. Therefore, to keep the original pointer size on 32-bit systems only pointer encryption and overflow detection are viable. Both solutions can be made context dependent, which is easily possible by using, for example, the current CFI state creating a strong synergy between previously discussed defense components. However, as the attacker can deduce the current CFI state, it necessary to introduce some kind of processor secret, so that the attacker cannot guess the canary or decrypt the pointer. An advantage of pointer encryption is that no memory overhead is created, as long as the cipher has the same block size as the pointer.

5 Our Recommendation

Based on the results of our literature study we now present our recommendation for a holistic ISA extension. We explain our approach by dividing programs into their individual levels mainly the basic block level, the functional level and the global level. For each level we name the main threat and show how it can be countered using our extension. Using a bottom-up approach, we show how we can reuse primitives from lower levels to facilitate other higher-level protection mechanisms.

5.1 Basic Block Level

We start by analyzing a single basic block, which on its own, consists only of sequence of instructions. On this level, an adversary may be interested in either changing the semantics of one or multiple instructions or skipping them entirely using glitches. In Sect. 4.1 we showed several defenses against glitch attacks.

In accordance with our discussion in Sect. 4.1, we propose the usage of *instruction hashing* as a countermeasure on this level. Compared to the other proposed solutions, hashing allows us to keep the original instructions unchanged and only add the operations necessary for the hashing process. Furthermore, the hash does not only give us information about the integrity of a basic block, but

also a value which characterizes the execution history, albeit in a limited form. This can later be used for different protection mechanisms. Thus, several additions have to be made to the processor: Firstly, we have to provide a hardware unit that is closely connected to the processor pipeline which is responsible for the hashing. Secondly, each basic block has to be augmented with a **CHECK** instruction, which embeds an expected hash which is compared to the actual computed hash. If they do not match, an error will be raised. The **CHECK** instruction can theoretically be placed arbitrarily, however, in this context only two positions make sense. Either the hash is checked at the end of a basic block or at the start. A position between the start and the end of a basic block would not cover the whole instruction sequence. We placed the **CHECK** instruction at the beginning of each block as shown in Fig. 1, so that it can be later used as a CFI mechanism. Therefore, in practice the integrity of basic block BB_t is checked in the following block BB_{t+1} . A straightforward approach for the hashing itself would be a simple XOR function, however, as pointed out by Werner *et al.* [28], a CRC function can provide better security by increasing the complexity for the attacker to mask his glitch attempt.

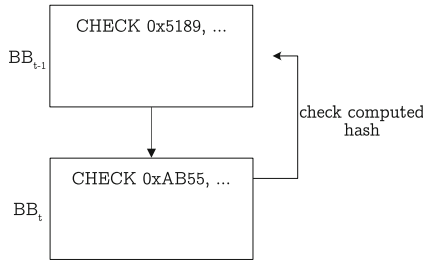


Fig. 1. The **CHECK** instruction provides a simple interface to determine the correctness of the previous basic block.

5.2 Function Level

A function is comprised of several basic blocks, however the execution path between them may not be linear but consist of more complex paths created, which may cause some blocks not to be executed. Here, an attacker can try to manipulate the intra-functional control flow, for example, by skipping branch instructions and causing a fall-through.

The **CHECK** instruction introduced in the previous section already implicitly provides a protection mechanism against intra-functional control flow violations. However, typical functions do usually not consist of a pure linear execution path but incorporate branches and loops. In this case, a simple scheme consisting of **CHECK** instructions is not sufficient, because the execution history differs for each side of a branch resulting in two different hashes. Thus, once the execution rejoins

the common path of the function, the `CHECK` instruction will fail. Consequently, we need to be able to synchronize the hashes. Therefore, if we identify a basic block with multiple predecessors, we select the hash of one predecessor as the expected hash and add a `CORRECT` instruction to each other predecessor, which synchronizes the hash to match the expected one. Similarly to other proposal, we may use the XOR function to correct the hash. An overview is given in Fig. 2.

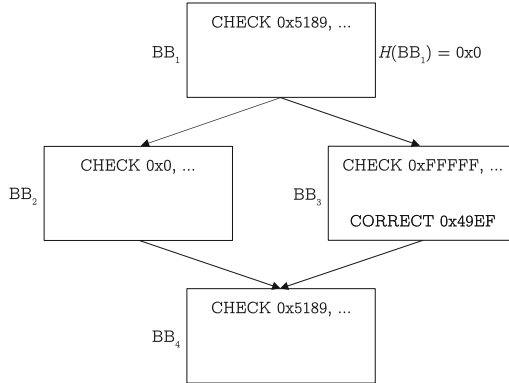


Fig. 2. Example of a `CORRECT` instruction being issued to synchronize the hashes. Note that BB_2 and BB_3 have inverted hashes to prevent a simple branch target change.

The above approach exhibits several weaknesses, which have to be addressed to make it suitable as an effective defense method. First, in case of a branch both successors of the basic block will necessarily have the same hash. Therefore, it may be possible to glitch the processor state and execute, for example, the *not taken* path instead the *taken* path. However, it is possible to diversify the hash by taking the *taken / not taken* decision into account when generating the hash. We want to note, that this may make hash synchronization more complicated in case of a complex control flow. This problem may be resolved by the compiler by inserting small trampolines including additional `CORRECT` instructions to properly synchronize the hash. Second, if the attacker can cause the execution of `CORRECT` instructions, he can change the hash state arbitrarily. In consequence, the placement of these instructions should be limited. The processor can be augmented with a Finite State Machine (FSM), which only allows corrections when in a specific state. From the previous description it is clear, that a correction should only be issued before a basic block ends. A basic block either ends in a fall-through, in which case the following instruction is a `CHECK`, or in a branch, which then it term is also followed by a `CHECK`. Enforcing these rules via a FSM restricts the attacker from freely executing corrections. However, there are several other attack vectors which can be closed by enforcing additional rules. As an example, an adversary can attempt to skip the branch at the end of a basic block and the following `CHECK` instruction, thus creating a linear control flow. This can

be counteracted by making the processor aware of the basic block length and throwing an exception if it does not encounter a CHECK after the specified length similarly to ISIS [20]. Here, a trade-off has to be made. Encoding the length into the CHECK instruction leads to a shorter hash length or scarifies other information bits, which potentially makes it easier to cause hash collisions. Instead, we propose to add an internal counter to the processor, which counts the instructions of each basic block and specifies a maximum amount. If the basic block does not reach its end before hitting the specified threshold, we assume that a manipulation took place and throw an error. This has the advantage, that we do not sacrifice encoding space and furthermore, the counter may be accessible to the programmer via fuse-bits to allow for device-specific modifications.

5.3 Global Level

On a global level a program consists of multiple functions. Here, well-known software-based attacks may take place, such as CRAs or code-injection attacks. Additionally, control-flow bending may be used to divert the execution along an alternative but valid path along the CFG. Thus, CFI as well as memory integrity as described in Sect. 4.2 and Sect. 4.3 are of most importance on this level.

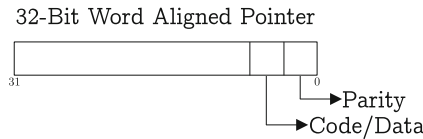


Fig. 3. Structure of a protected pointer exploiting the aligned property to store additional information.

The included CHECK and CORRECT instructions already provide an CFI protection mechanism. Each hash can be treated as the label of a specific basic block or function. The coarseness of this CFI protection is dependent on the hash length. For example, an 8-bit hash only allows for 256 values. It is safe to assume, that complex embedded software consists of more than 256 basic blocks, which will inevitably lead to some form of CFG relaxation. This means, that multiple blocks will be assigned the same label. Therefore, the control flow can be diverted to any basic block which shares the label of the original destination. Even with larger hash sizes, this can become problematic. According to our previously explained mechanism, a synchronization has to take place at the source locations and/or at the end of the target function. Therefore, all source locations create the same target label using corrections and the function creates one label which is shared between all return destinations. To protect against this threat we propose the introduction of a *pointer encryption* scheme to hinder the attacker from injection valid return addresses. *Pointer Authentication* akin to

ARM’s PACs are only feasible on systems with unused address bits. On embedded systems, the predominate architecture is however 32-bits of which almost all bits are required. However, we can combine elements of both approaches to facilitate a mechanism which can be used in the same way as ARM’s PAC. Whenever a function is entered, we encrypt the return address in the link register using a key comprised of the expected hash at the usage location and an internal secret, which may be a random number generated during the boot process of the microcontroller. Then, at the intended usage location, e.g. the end of the function, we decrypt the return address using the hash of the current location and the internal secret. Only if the hashes match, the correct decryption key is generated. Therefore, the attacker has no direct control over pointer. Under the assumption of an architecture which only allows word-aligned accesses, this mechanism can be enhanced even further. In this case, the two least significant bits are always zero. Therefore, we can encode further information into the pointer as shown in Fig. 3. One bit can be used to indicate whether the pointer refers to code or data and the other bit can be used to perform a simple parity check. This allows programs to differentiate between pointer types and create secured data and code pointers which are only usable at their intended locations similar to ARM PAC. The protection mechanism can be seen in Fig. 4.

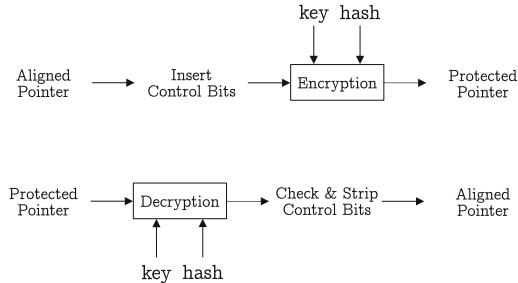


Fig. 4. Overview of the pointer protection process.

Furthermore, we can enforce some stricter CFI rules using our FSM and additional information in each CHECK instruction as seen in Fig. 5. We can use one bit each to encode whether the basic block is an function entry, which prevents targeting arbitrary basic blocks using a call, and we can mark all basic blocks which are a return target. Thus, disallowing arbitrary returns to basic blocks which are not actual return locations.

6 Proof-of-concept

We now demonstrate the practicability of our approach by implementing the proposed instruction set extension design for *RISC-V* and evaluate it using the cycle-accurate *gem5* simulator [17].

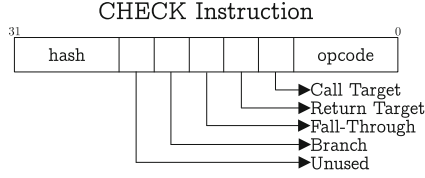


Fig. 5. Internals of the CHECK instruction showing the hash as well as bitfields specifying basic block specifics.

Table 1. Instructions required to implement our recommendation.

Instruction	Operands	Description
CHECK	20-bit hash, 5-bit properties	Checks hash and defines properties of current block
CORRECT	20-bit value	Performs a correction to synchronize hash
ENCCPTR	20-bit hash, register	Generates a code pointer out of aligned pointer and encrypts it
ENDDPTR	20-bit hash, register	Generates a data pointer out of aligned pointer and encrypts it
DECCPTR	register	Decrypts pointer and strips code pointer data
DECDPTR	register	Decrypts pointer and strips data pointer data

6.1 ISA Implementation and Simulation

We chose the RISC-V 32-bit standard as our target ISA because of its built-in support for user-defined instructions and its rising popularity in embedded systems. In total, we require six new instructions: two instructions to implement anti-glitching and CFI as well as four instructions to handle the pointer protection. Furthermore two new registers are added which hold the current hash state and the processor secret, see Table 1 for an overview of the additional instructions. The hash length was set to 20 bits, so that the CHECK instruction is able to hold the hash as well as state information for each basic block. We added support for these custom instructions and registers to the *gem5* simulator. Note that at the time of writing, *gem5* only supports the 64-bit RISC-V standard, however, all 32-bit opcodes stay valid, thus allowing us to simulate 32-bit code. Furthermore, we also added the required FSM and encryption/decryption module to the simulated processor to enforce the rules outlined in Sect. 5.2 as well as Sect. 5.3 and to enable support for the pointer encryption. As we operate on 32-bit pointers, we chose the hardware-optimized *SIMON* cipher [4] with a block length of 32 bits and a key length of 64 bits. The encryption uses a connotation of the expected hash with a 44-bit processor secret to form the 64-bit key.

6.2 Code Transformation

Our ISA extension can be implemented as a compiler extension or via a binary translation approach. For this work, we chose the latter approach. We achieve this by using a proprietary binary transformation framework that is capable of analyzing and manipulating existing binaries. The framework first lifts the code into an intermediate language. On this abstraction level, we analyze the control

flow of the program. Furthermore, we are able to add instructions without being constrained by the basic block placement in the memory layout. During code generation, the intermediate language instructions are resolved into a memory layout and adapted to the updated basic block locations. The transformation itself takes place in 3 passes. In the first pass, we add an empty `CHECK` instruction to each basic block. During this pass we identify functions for the return address protection. However, other code or data pointers are not automatically protected and have to be secured manually by the programmer. The second pass simulates the hashing process and identifies possible conflicts, i.e. a basic block is targeting another block which already has an assigned hash resolves them by adding corrections. The third pass performs the actual hashing and inserts the expected values and correction values in the empty placeholder instructions thus creates the finalized program. During this process it may happen, that the framework requires a yet not computed hash, for example, during the pointer encryption and decryption process. Therefore, the framework picks a random known hash for the target location and issues a correction before the decryption takes place. We modified the FSM to cover this case.

Our chosen approach naturally exhibits some weaknesses. For example, during automated program analysis it may not be possible to extract all targets of an indirect call. However, we assume that the benign user wants to protect software. Therefore, the call graph information can be provided as an additional input to the binary transformation framework to resolve such issues. Note that for general-purpose software and a benign user, this problem does not occur if a compiler-based approach is chosen to arm the software with our instruction set extension.

6.3 Software and Hardware Considerations

Until now, we only considered the protection of a single program. However, a complex embedded system may execute multiple processes with shared libraries and interrupts. Therefore, some modifications would have to be done to the software and hardware. In case a (real-time) operating system is used, the context switch has to be altered to include the internal hash register, as the hash state differs for each program. To increase the security, the processor secret may also be different for each process, which would require saving the key register as well. On the hardware side, the unit responsible for saving the execution state when entering an interrupt also has to store the internal hash register (and key register if desired). Additionally, an encryption and hash unit have to be integrated into the processor. To achieve a high throughput both should include an unrolled implementation, which trades space overhead for a shorter execution time. It should be noted, that the hashing can be performed in parallel instead of being a pipeline stage.

6.4 Security Evaluation

In the following section, we systematically evaluate the security of our proposed solution by discussing each potential attack vector and how our design defends against it.

Glitching Attacks. Our hashing approach decreases the success probability of a glitching attack by introducing regular checks into the instruction stream. As it is improbable that the hash will still match the expected value when skipping or glitching a single instruction, the glitch will be detected with a latency of $t - i$ where t is the length of the basic block and i the index of the glitched instruction within the block. To circumvent the detection, the attacker is forced to perform multiple glitches. Additionally, the choice of the hash function can restrict the attacker even further as shown by Werner *et al.* [28], because the subsequent bitflips required to hide a fault may be located far apart making it likely that a check happens before a correction is possible.

Control-Flow Attacks. The proposed solution can deter glitching-based control-flow attacks as well as traditional software-based attacks. In the first case, the attacker may try to change the target of a branch by forcing a fall through by glitching the processor state. However, by letting the branch result influence the expected state and by introducing a counter, these kind of attacks are severely hampered. For software-based attacks, the gadget-space is drastically reduced, as the return target must be a basic block entry point. Chaining such relatively long instruction sequences together will induce various side-effects. Furthermore, the hash state must match and is influenced by the executed basic blocks. RISC-V specific ROP attacks demonstrated by Jaloyan *et al.* [14] which abuse the mixing of compressed and uncompressed instructions are also unfeasible, as they alter the hash state. Control-flow bending stays possible, as the hash space is limited. However, by employing our pointer protection, the attacker cannot arbitrarily change the return address. Under the assumption that an encryption gadget does not exist, the attacker cannot create his own protected pointers. Instead, he would require an information leak and a write gadget to possibly exchange the return address for another valid return address. Table jumps form a special case, because all targets share the same hash and our pointer protection cannot be applied. Here, special precautions have to be made by the programmer in software. However, arbitrary jumps stay impossible as the target must be designated as a jump target.

Memory Attacks. Our extension only partially covers memory attacks, as we do not prevent writing or reading from memory. Instead, by using our pointer protection scheme we can stop buffer overflows from being an attack vector. On its own a buffer overflow or any other writing gadget only present a risk, if the attacker is able to overwrite control-flow critical data.

Table 2. Size and runtime overhead created by our recommendation when compiling with `-Os`.

Benchmark	Size Overhead	Execution Overhead
Adcpm_encode	37.50%	43.64%
Aes	19.86%	21.87%
Blowfish	18.18%	11.60%
Crc	30.51%	45.60%
Fft	26.32%	21.45%
Rsa	21.71%	26.89%
Sha	19.12%	24.94%
Average	20.13%	28.00%

6.5 Evaluation

To evaluate our solution, we chose the MiBench2 [18] benchmark suite, which contains various workloads that are commonly found on embedded devices. We compiled them using the `-Os` compile flag to create space efficient code. As we are only interested in overhead produced both in the binary and during the execution, we chose to run our *gem5* implementation using the *Atomic-SimpleCPU* model, which simulates a rudimentary single-issue processor, and the *System Emulation* mode to easily load and execute binaries. The results of our evaluation can be seen in Table 2. On average we observe a size overhead of 20.13% and an execution overhead of 28.00%. We note that the overhead created is highly dependent on the size of the basic blocks, as one instruction is added at least for each basic block with the possibility of an additional correction instruction. Small basic blocks will lead to a high amount of additional instructions, whereas large basic blocks only require few additional instructions but may give the attacker the possibility to mask his glitch attempt. Therefore, a trade-off must be found. We measured the average basic block length of the `aes` benchmark using different compiler optimizations and found that `-O1`, `-O2` and `-Os` produce smaller basic blocks with an average length of 8 instructions. On the contrary, `-O0`, so no optimization, as well as `-O3` produce larger basic blocks with an average size of 17 instructions. Consequently, protecting the `aes` benchmark compiled using `-O3` only results in a size overhead of 9.15% and a execution overhead of 2.72%.

Several works exist which implement some parts of our solution. For example, the authors of ISIS [20] add to a control word to each basic block, state a memory overhead of 12% to 15%. However, ISIS misses some protection mechanisms such as the pointer protection and can also not cover all possible branches. The work by Werner *et al.* [27], is the only work at the time of writing, which tries to achieve a holistic protection against faults. However, their protection is based on multiple approaches which do not build upon each other. A mixture of instruction encryption, branch encoding as well as pointer protection is used

to protect against glitches. The authors state an average size overhead of 19.8% for the instruction encryption and an average runtime overhead of 9.1%. The branch encoding occurs an additional size overhead of 2.5%. Lastly the pointer protection costs on average 9.99% in binary size and 6.34% in runtime. Thus, a combination of their approaches results in a higher size overhead compared to our solution, whereas our extension increases the amount of executed instructions. Furthermore, a downside of their approach is the encryption process, which is costly and may lead to unpredictable behaviour in case of a glitch, as the decryption may result in a valid but random instruction.

7 Future Work

In this work, we focused on developing a holistic secure ISA extension and implemented it in a simulated environment to determine its overhead. Future work may explore how to implement our recommendation in hardware and analyze its effectiveness by performing clock and voltage glitches. We want to emphasize that this research direction is especially relevant in the context of silicon root-of-trust where a glitch-induced instruction skip can have severe consequences, i.e. to manipulate a secure boot verification check.

8 Conclusion

The rising popularity of resource-constrained embedded devices in security-relevant areas increases the necessity for efficient and effective countermeasures, in particular to protect against attackers with physical access. In this work, we systematically analyzed the state-of-the-art for fault attacks and software-based attacks on embedded systems. We then discussed defense technique combinations and designed a novel approach to form a small instruction set extension that exhibits effective security against the aforementioned threats. We then implemented our approach using binary translation to arm software with our instruction set extension and then evaluated our approach using *gem5* and *MiBench2*. In summary, our results showed that our extension is a competitive candidate for a holistic secure embedded ISA extension.

Acknowledgments. We would like to thank our anonymous reviewers for their constructive feedback. The work described in this paper has been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) through the project RAINCOAT (440059533), by the DFG through Germany’s Excellence Strategy - EXC 2092 CASA - 390781972, and by the German Federal Ministry of Education and Research (BMBF) through the project FlexKI (01IS220861).

References

1. ARM: Armv8.5-A Memory Tagging Extension White Paper. ARM (2019)
2. Balasch, J., Gierlichs, B., Verbauwhede, I.: An in-depth and black-box characterization of the effects of clock glitches on 8-bit MCUs. In: Breveglieri, L., Guilley, S., Koren, I., Naccache, D., Takahashi, J. (eds.) 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, pp. 105–114. IEEE Computer Society, Tokyo, Japan (2011). <https://doi.org/10.1109/FDTC.2011.9>
3. Barry, T., Couroussé, D., Robisson, B.: Compilation of a countermeasure against instruction-skip fault attacks. In: Palkovic, M., Agosta, G., Barenghi, A., Koren, I., Pelosi, G. (eds.) In: Proceedings of the 3rd Workshop on Cryptography and Security in Computing Systems, CS2@HiPEAC, Prague, pp. 1–6. ACM Czech Republic (2016). <https://doi.org/10.1145/2858930.2858931>
4. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. IACR Cryptol. ePrint Arch., p. 404 (2013). <http://eprint.iacr.org/2013/404>
5. Bradbury, A., Ferris, G., Mullins, R.: Tagged memory and minion cores in the lowRISC SoC. University of Cambridge, Memo (2014)
6. Burow, N., Zhang, X., Payer, M.: Shining light on shadow stacks. CoRR abs/1811.03165 (2018). <http://arxiv.org/abs/1811.03165>
7. Christoulakis, N., Christou, G., Athanasopoulos, E., Ioannidis, S.: HCFI: hardware-enforced control-flow integrity. In: Bertino, E., Sandhu, R.S., Pretschner, A. (eds.) Proceedings of the 6th ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, pp. 38–49. ACM New Orleans, LA, USA (2016). <https://doi.org/10.1145/2857705.2857722>
8. de Clercq, R., et al.: SOFIA: software and control flow integrity architecture. In: Fanucci, L., Teich, J. (eds.) 2016 Design, Automation Test in Europe Conference Exhibition, DATE, pp. 1172–1177. IEEE 2016, Dresden, Germany (2016). <https://ieeexplore.ieee.org/document/7459489/>
9. de Clercq, R., Verbauwhede, I.: A survey of hardware-based control flow integrity (CFI). CoRR abs/1706.07257 (2017). <http://arxiv.org/abs/1706.07257>
10. Cowan, C.: StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks. Rubin, A.D. (ed.) In: Proceedings of the 7th USENIX Security Symposium, 98, P. 5–5 San Antonio, TX, USA, USENIX Association (1998). <https://www.usenix.org/conference/7th-usenix-security-symposium/stackguard-automatic-adaptive-detection-and-prevention>
11. Davi, L., et al.: HAFIX: hardware-assisted flow integrity extension. In: Proceedings of the 52nd Annual Design Automation Conference, pp. 741–746 ACM, San Francisco, CA, USA (2015). <https://doi.org/10.1145/2744769.2744847>
12. De, A., Basu, A., Ghosh, S., Jaeger, T.: Hardware assisted buffer protection mechanisms for embedded RISC-V. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **39**(12), 4453–4465 (2020). <https://doi.org/10.1109/TCAD.2020.2984407>
13. Fei, Y., Shi, Z.J.: Microarchitectural support for program code integrity monitoring in application-specific instruction set processors. In: Lauwereins, R., Madsen, J. (eds.) 2007 Design, Automation and Test in Europe Conference and Exposition, DATE 2007, pp. 815–820. EDA Consortium, San Jose, Nice, France, CA, USA (2007). <https://doi.org/10.1109/DATE.2007.364391>

14. Jaloyan, G., Markantonakis, K., Akram, R.N., Robin, D., Mayes, K., Naccache, D.: Return-oriented programming on RISC-V. In: Sun, H., Shieh, S., Gu, G., Ate-niese, G. (eds.) ASIA CCS '20: The 15th ACM Asia Conference on Computer and Communications Security, Taipei, Taiwan, pp. 471–480. ACM (2020). <https://doi.org/10.1145/3320269.3384738>
15. Kayaalp, M., Schmitt, T., Nomani, J., Ponomarev, D., Abu-Ghazaleh, N.B.: SCRAP: architecture for signature-based protection from code reuse attacks. In: 19th IEEE International Symposium on High Performance Computer Architecture, HPCA 2013, pp. 258–269. IEEE Computer Society Shenzhen, China (2013). <https://doi.org/10.1109/HPCA.2013.6522324>
16. Kim, H., Lee, J., Pratama, D., Awaludin, A.M., Kim, H., Kwon, D.: RIMI: instruction-level memory isolation for embedded systems on RISC-V. In: IEEE/ACM International Conference on Computer Aided Design, ICCAD 2020, pp. 341–349. IEEE San Diego, CA, USA (2020). <https://doi.org/10.1145/3400302.3415727>
17. Lowe-Power, J., et al.: The gem5 simulator: Version 20.0+. CoRR abs/2007.03152 (2020). <https://arxiv.org/abs/2007.03152>
18. Mibench2 (2022). <https://github.com/impedimentToProgress/MiBench2>
19. Ohlsson, J., Rimén, M., Gunneflo, U.: A study of the effects of transient fault injection into a 32-bit RISC with built-in watchdog. In: Digest of Papers: FTCS-22, The 22nd Annual International Symposium on Fault-Tolerant Computing, Boston, Massachusetts, pp. 316–325. USA IEEE Computer Society (1992). <https://doi.org/10.1109/FTCS.1992.243569>
20. Rodríguez, F., Campelo, J., Serrano, J.J.: A Watchdog Processor Architecture with Minimal Performance Overhead. In: Anderson, S., Felici, M., Bologna, S. (eds.) SAFECOMP 2002. LNCS, vol. 2434, pp. 261–272. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45732-1_26
21. Rodríguez, F., Serrano, Juan J.: Control Flow Error Checking with ISIS. In: Yang, L.T., Zhou, X., Zhao, W., Wu, Z., Zhu, Y., Lin, Man (eds.) ICESS 2005. LNCS, vol. 3820, pp. 659–670. Springer, Heidelberg (2005). https://doi.org/10.1007/11599555_63
22. Savry, O., El-Majihi, M., Hiscock, T.: Confidaent: control flow protection with instruction and data authenticated encryption. In: 23rd Euromicro Conference on Digital System Design, pp. 246–253. IEEE DSD 2020, Kranj, Slovenia, (2020). <https://doi.org/10.1109/DSD51259.2020.00048>
23. Security, Q.P.: Pointer Authentication on ARMv8.3 - Design and Analysis of the New Software Security Instructions. Qualcomm Technologies, Inc. (2017)
24. Selmke, B., Hauschild, F., Obermaier, J.: Peak clock: Fault injection into PLL-Based systems via clock manipulation. In: Chang, C., Rührmair, U., Holcomb, D.E., Schaumont, P. (eds.) In: Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop, ASHES@CCS 2019, pp. 85–94. ACM London, UK, (2019). <https://doi.org/10.1145/3338508.3359577>
25. Shanbhogue, V., Gupta, D., Sahita, R.: Security analysis of processor instruction set architecture for enforcing control-flow integrity. In: Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy, HASP@ISCA 2019, pp. 801–811. ACM (2019). <https://doi.org/10.1145/3337167.3337175>
26. Spensky, C., et al.: Glitching demystified: Analyzing control-flow-based glitching attacks and defenses. In: 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2021, pp. 400–412. IEEE Taipei, Taiwan, (2021). <https://doi.org/10.1109/DSN48987.2021.00051>

27. Werner, M., Schilling, R., Unterluggauer, T., Mangard, S.: Protecting RISC-V processors against physical attacks. In: Teich, J., Fummi, F. (eds.) Design, Automation Test in Europe Conference Exhibition, DATE 2019, pp. 1136–1141. IEEE Florence, Italy (2019). <https://doi.org/10.23919/DATE.2019.8714811>
28. Werner, M., Wenger, E., Mangard, S.: Protecting the Control Flow of Embedded Processors against Fault Attacks. In: Homma, N., Medwed, M. (eds.) CARDIS 2015. LNCS, vol. 9514, pp. 161–176. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31271-2_10
29. Wilken, K.D., Shen, J.P.: Continuous signature monitoring: Efficient concurrent-detection of processor control errors. In: Proceedings International Test Conference, pp. 914–925. IEEE Computer Society. Washington, D.C., USA, (1988). <https://doi.org/10.1109/TEST.1988.207880>
30. Witteman, M., Oostdijk, M.: Secure application programming in the presence of side channel attacks. In: RSA conference. (2008)
31. Woodruff, J., et al.: CHERI concentrate: Practical compressed capabilities. IEEE Trans. Computers **68**(10), 1455–1469 (2019). <https://doi.org/10.1109/TC.2019.2914037>
32. Woodruff, J., et al.: The CHERI capability model: Revisiting RISC in an age of risk. In: ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, pp. 457–468. IEEE Computer Society. Minneapolis, MN, USA, (2014). <https://doi.org/10.1109/ISCA.2014.6853201>
33. Yuce, B., Ghalaty, N.F., Deshpande, C., Patrick, C., Nazhandali, L., Schaumont, P.: FAME: fault-attack aware microprocessor extensions for hardware fault detection and software fault response. In: Proceedings of the Hardware and Architectural Support for Security and Privacy 2016, HASP@ICSA, pp. 81–88. ACM Seoul, Republic of Korea (2016). <https://doi.org/10.1145/2948618.2948626>
34. Zhu, G., Tyagi, A.: Protection against indirect overflow attacks on pointers. In: Cole, J.L., Wolthusen, S.D. (eds.) In: Proceedings of the 2nd IEEE International Workshop on Information Assurance (IWIA'04), pp. 97–106. IEEE Computer Society Charlotte, North Carolina, USA(2004). <https://doi.org/10.1109/IWIA.2004.1288041>



QuantumCharge: Post-Quantum Cryptography for Electric Vehicle Charging

Dustin Kern¹(✉), Christoph Krauß¹, Timm Lauser¹, Nouri Alnahawi¹, Alexander Wiesmaier¹, and Ruben Niederhagen^{2,3}

¹ Darmstadt University of Applied Sciences, Darmstadt, Germany
{dustin.kern, christoph.krauss, timm.lauser, nouri.alnahawi, alexander.wiesmaier}@h-da.de

² University of Southern Denmark, Odense, Denmark
ruben@polycephaly.org

³ Academia Sinica, Taipei, Taiwan

Abstract. ISO 15118 enables charging and billing of Electric Vehicles (EVs) without user interaction by using locally installed cryptographic credentials that must be secure over the long lifetime of vehicles. In the dawn of quantum computers, Post-Quantum Cryptography (PQC) needs to be integrated into the EV charging infrastructure. In this paper, we propose QuantumCharge, a PQC extension for ISO 15118, which includes concepts for migration, crypto-agility, verifiable security, and the use of PQC-enabled hardware security modules. Our prototypical implementation and the practical evaluation demonstrate the feasibility, and our formal analysis shows the security of QuantumCharge, which thus paves the way for secure EV charging infrastructures of the future.

Keywords: Post-Quantum Cryptography · Electric Vehicle Charging · Formal Security Verification · ISO 15118 · Hardware Security Module

1 Introduction

One of the driving technologies of the 21st century is going to be quantum computing. However, while quantum computing promises great impact, e.g., in the fields of chemical and biological engineering, artificial intelligence, financial services, and complex manufacturing [15], it also poses a severe threat to our current IT security. While current prototypes of quantum computers are not yet large and stable enough to pose an immediate threat, experts predict that in the upcoming years practical attacks are becoming more and more likely [48]. For instance, Shor's algorithm [58] provided that it is executed on a sufficiently large and stable quantum computer, can break the asymmetric cryptography that is currently in wide use — RSA, DSA, DH, and ECC. This will affect all of our current IT infrastructures including automotive protocols.

In recent years, the term Post-Quantum Cryptography (PQC) has been established for the next generation of cryptography providing protection against crypt-analytic attacks aided by large quantum computers. Much effort is taken to develop and integrate PQC [3], the most prominent of which is the ongoing standardization effort [50] of the National Institute of Standards and Technology (NIST) where the first candidates for standardization were stipulated in July 2022. However, PQC's requirements regarding computational power, storage, and memory pose challenges for its use in resource-restricted devices [9].

With the long lifespan of entities in the e-mobility context, such as 10+ years for Charge Points (CPs) [60] or up to 35 years for Electric Vehicles (EVs) [37], considerations of PQC and crypto-agility are critical in order to maintain security. One of the most important e-mobility standards is ISO 15118 [36,37], which defines a Plug-and-Charge (PnC) protocol enabling charging and billing of EVs based on cryptographic credentials installed directly in the car making RFID cards or apps obsolete. An integration of PQC schemes into ISO 15118, however, is not straightforward. First, several entities defined in this standard are embedded devices and as such subject to strict resource restrictions hampering the integration of PQC. Second, ISO 15118 is not crypto-agile, i.e., it does not provide a mechanism to add and agree upon new cryptographic schemes, which requires changes to the standard when introducing PQC. Instead, it specifies hard requirements for the used cryptography (such as specific algorithms and parameters or limited execution times and data sizes) due to interoperability reasons.

Related Work. An overview of the challenges and the current state of migrating applications and communication protocols to PQC is provided in [3]. Different platforms and use cases impose different requirements w.r.t. the choice of a suitable PQC scheme and parameters as discussed in [9,53]. PQC schemes are benchmarked and tailored to specific hardware and software platforms in [12,21]. Other works focus on special requirements for embedded/resource-constrained devices [5,44] or integrating PQC in micro-controllers [31,52]. Efforts for integrating PQC in prominent protocols include IKEv2 [61] and VPN [33,45]. There is also work on integration/migration strategies [9,23] and crypto-agility [2,43]. The integration into the industrial protocol Open Platform Communications Unified Architecture (OPC UA) has been discussed in [55] for cyber-physical systems.

A major focus has been given to PQC for Transport Layer Security (TLS): Several experiments and approaches are presented in the literature for integrating PQC into TLS including benchmarks [54,59], hybrid approaches [20,62], and investigations for embedded systems [16,18]. The Open Quantum Safe (OQS) project provides a collection of implementations of several PQC schemes in the library *liboqs* and integration into several popular Internet protocols including forks of the BoringSSL and OpenSSL libraries with the integration of PQC using *liboqs* [63].

The suitability of NIST candidates for Vehicle to Vehicle communication is assessed in [13] and [30] provides a light-weight identity-based two-party Authen-

ticated Key Exchange for the Internet of vehicles. A case study of PQC for secure communication within a vehicle is given in [17] and a concrete implementation is evaluated for the protocol Lightweight Authentication for Secure Automotive Networks (LASAN) in [56]. Optimizations for PQC in automotive systems are provided in [25]. [67] provides an analysis of NIST PQC candidates for usage in Hardware Security Modules (HSMs) and proposes new sets of hardware accelerators for the future generation of the automotive HSMs.

To the best of our knowledge, no related work focuses on PQC security in EV charging protocols. However, some papers investigate the use of HSMs with ISO 15118. In [27], an approach is presented that secures an EV's PnC credentials, which are per ISO 15118 definition generated in the backend, by importing them into the EV's HSM (specifically a Trusted Platform Module (TPM) 2.0). Notably, the approach of [27] has been integrated into the current draft of the upcoming second edition of the standard, ISO 15118-20 [37]. However, the generation of private credentials in the backend results in a needlessly high risk of backend leaks. This issue is addressed in [26, 28], where private keys are generated within the EV's HSM (specifically a TPM 2.0) and the EV only request a corresponding certificate from the backend. Critical aspects that are not considered in this context include: (i) crypto-agility, (ii) PQC-support including migration, and (iii) formal security proofs.

Our Contribution. We propose QuantumCharge, an ISO 15118 protocol extension which integrates crypto-agility and the support of PQC into the EV charging and billing process. This paper has the following eight main contributions: (i) We analyze the cryptographic algorithms used in the standards w.r.t. their security against attacks from quantum computers and propose alternatives following the NIST standardization. (ii) We propose the QuantumCharge architecture including a crypto-agility concept which supports multiple cryptographic algorithms (including PQC) and extends the ISO 15118 process to support algorithm negotiation. This enables switching to other algorithms if an algorithm proves to be insecure. (iii) We support migration to address the issue of using existing legacy PnC entities that cannot be updated to support QuantumCharge. (iv) We have designed QuantumCharge in such a way that the desired strong security properties can be shown with symbolic proofs. (v) We propose a concept for integrating PQC-enabled HSMs to ensure secure key generation, storage, and usage as well as other security services such as secure boot. (vi) We provide an implementation and performance evaluation showing the feasibility of our approach and compliance with ISO 15118 limitations such as timeouts. (vii) We perform a formal analysis of the central aspects of our protocol extension using the Tamarin prover to show the security of QuantumCharge. (viii) We release the adapted ISO 15118 implementation and Tamarin models as open source (cf. Sects. 6 and 7).

Structure of the Paper. The remainder of the paper is structured as follows. Section 2 provides background on e-mobility and PQC on embedded systems. Section 3 introduces our assumed system and attacker model and Sect. 4 the requirements for QuantumCharge. Section 5 describes our QuantumCharge con-

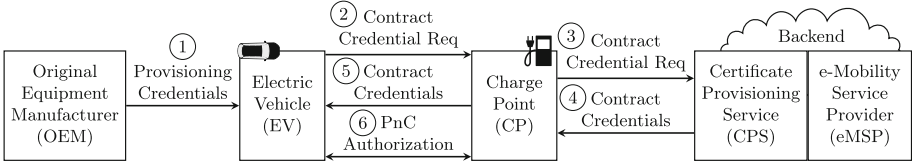


Fig. 1. E-mobility architecture.

cept. Our formal security evaluation is described in Sect. 6 and the implementation and practical evaluation in Sect. 7. Finally, we conclude the paper in Sect. 8.

2 Background

In this section, we introduce background on e-mobility and relevant standards as well as PQC and its performance on embedded systems.

2.1 E-Mobility Architecture

Figure 1 shows a simplified e-mobility architecture. It adopts the definitions of the ISO 15118 standard [35,36] for entities and processes related to PnC authentication for EV charging. The entities include the EV, the CP, and several PnC backend systems. For credential management, only the EV’s Original Equipment Manufacturer (OEM), the e-Mobility Service Provider (eMSP) backend, and the Certificate Provisioning Service (CPS) are relevant. The EV user has a contract with the eMSP, which enables PnC authorization and billing based on cryptographic credentials. The CPS establishes trust in the credentials provided by an eMSP for an EV. The EV stores the required credentials and establishes a PnC session with the CP. The CP (managed by a Charge Point Operator; not shown) enables data transfer to the backend and authorizes the EV to use the charging service.

The ISO 15118 communication between EV and CP uses a TLS channel with unilateral authentication of the CP. The EV authenticates itself inside the TLS channel using a challenge-response protocol. In the upcoming second edition, ISO 15118-20 [37], TLS uses mutual authentication with a vehicle certificate installed by the OEM in addition to the challenge-response-based EV authentication within the TLS channel. ISO 15118-20 further includes more modern TLS cipher suites, the option for TPM-based credential protection, and backward compatibility. The communication between CPs and the backend uses different protocols, e.g., Open Charge Point Protocol (OCPP) [51], and is usually secured with TLS. We omit the details for the sake of simplicity. In the following, we describe the relevant steps for using PnC with a focus on the related application-layer security mechanisms.

EV Preparation. During production, the OEM generates the EV's provisioning credentials (Step 1 in Fig. 1) in the OEM backend and installs them in the EV. The credentials consist of a unique long-term identity called Provisioning Certificate Identifier (PCID), the key pair $\{PC_{pk}; PC_{sk}\}$,¹ and the respective X.509 *OEM Provisioning Certificate* PC_{Cert} (which includes PCID and PC_{pk}). The key pair $\{PC_{pk}; PC_{sk}\}$ is later used for cryptographic signatures (signature generation with PC_{sk} and verification with PC_{pk}) and for Elliptic Curve Diffie Hellman (ECDH)-based key exchange (using PC_{sk} on the EV's side and PC_{pk} on a secondary actor). In case the second edition of ISO 15118 is supported, the vehicle certificate and the corresponding private key are additionally generated and installed in the EV.

Generation of Contract Credentials. To enroll the EV for PnC charging services, the owner hands over the PCID to the eMSP when concluding a charging contract. The eMSP then generates contract credentials to enable the EV to authenticate itself against the CP. The eMSP generates a key pair $\{CC_{pk}; CC_{sk}\}$ and the X.509 *Contract Certificate* CC_{Cert} that contains a unique e-Mobility Account Identifier (eMAID) linked to the charging contract. The contract credential key pair $\{CC_{pk}; CC_{sk}\}$ can later be used by the EV for authentication via cryptographic signatures (signature generation with CC_{sk} and verification with CC_{pk}).

Contract Credentials Installation/Update. An EV's contract certificate CC_{Cert} and the respective key pair are installed when connecting the EV to a CP for the first time. First, a TLS channel is established (not shown in Fig. 1). In Edition 1 of ISO 15118, only the CP authenticates itself in the TLS handshake by using the private key corresponding to the Supply Equipment Communication Controller (SECC) certificate. The additional EV authentication is only supported in Edition 2 based on the vehicle certificate. After the TLS channel has been established, the EV sends a request to the CP to install the contract credentials (Step 2). The request contains PC_{Cert} and is signed with PC_{sk} . The CP forwards the request over the CPS to the eMSP (Step 3). The eMSP encrypts the private key CC_{sk} with PC_{pk} (from PC_{Cert}).² The CPS appends its certificate chain to the response data, signs the response, and sends it (Step 4) over the CP to the EV (Step 5). The EV first verifies the authenticity of the received data by validating the CPS' chain up to a locally installed root certificate and verifying the CPS' signature. Afterwards, the EV decrypts CC_{sk} and stores the credentials for later use. Credential updates use a similar process, except that old contract credentials are used for signing/encryption instead of the provisioning credentials.

¹ Note regarding notation: in the key pair $\{X_{pk}; X_{sk}\}$ we use X_{pk} to denote the public part of the key pair and X_{sk} for the secret part of the key pair.

² The encryption is done symmetrically using Advanced Encryption Standard (AES) for which a symmetric session key is generated with an ephemeral-static ECDH key exchange using PC_{pk} as the static part on the eMSP's side (and PC_{sk} as the static part on the EV's side).

Contract Credentials Usage. To start charging, the customers plug the charging cable of their EV into a CP, and the charging process is automatically handled with ISO 15118 without further customer interaction. Again, a TLS channel is established first. Then, the EV uses the contract credentials to authenticate its concluded contract against the CP, which authorizes the charging session (Step 6). The EV provides its eMAID and contract certificate CC_{Cert} to the CP. The CP verifies the validity of the credentials and sends a random nonce to the EV, which in turn signs the nonce with the private key CC_{sk} and returns the signature to the CP. If the authentication succeeds, the CP activates charging. Afterwards, the EV can periodically sign meter readings with its private key CC_{sk} to confirm the status of the charging session (see [36], Sect. 8.4.3.13). Before billing, signatures regarding the metering may be verified by eMSPs.

2.2 Post-Quantum Cryptography on Embedded Systems

In 2016, NIST started a PQC standardization process [50] that aims to standardize schemes for key encapsulation and signing, which were selected from five main families of PQC algorithms: code-based, hash-based, isogeny-based, lattice-based, and multivariate schemes. We focus on signatures (with key generation in the HSM, cf. Sect. 5). After several rounds, three signature-scheme candidates were selected for standardization in July 2022 [49]: The two lattice-based schemes CRYSTALS-Dilithium [7] and FALCON [24] as well as the hash-based scheme SPHINCS+ [6].

Notably, IETF already has standardized the “stateful” hash-based PQC signature schemes XMSS [32] and LMS [46]. Compared to the “stateless” algorithm SPHINCS+, handling of the state in XMSS and LMS requires additional precautions along with existing key-handling practices. While in a vehicle equipped with a TPM the handling of a state can be realized fairly safely and easily (e.g., using a monotonic counter in the TPM), more effort may be required for backend servers without TPM or HSM — e.g., backup and workload-sharing strategies need to be adapted to take state management into account. Due to these additional requirements specific to stateful schemes, we do not investigate XMSS and LMS in further detail. However, since SPHINCS+ generally requires more computational resources and has larger signatures than XMSS and LMS, SPHINCS+ is a “worst case” hash-based signature scheme and the results for SPHINCS+ also apply as an upper limit to XMSS and LMS.³

³ In addition, XMSS and LMS require the selection of parameters that define the maximum number of signatures per public-private key pair. These parameters can be chosen large enough to support the total number of expected charging operations during the lifetime of a vehicle. Given a maximum lifespan of 35 years for an electric vehicle and at most two charging operations per day, a maximum of 2^{20} signatures would provide a significant margin. Nevertheless, procedures for re-keying must be put in place when using XMSS and LMS for this application.

Performance benchmarks on an embedded Arm Cortex-A53 platform⁴ indicate that, in general, Dilithium stands out for its efficient key generation, signing, and verification [44]. FALCON is less efficient for signing and verification, yet still acceptable; however, key generation times are rather long. SPHINCS+ is considered the least efficient, with acceptable key generation and verification but extremely long signing times. Benchmarks of the pqm4 project [38, 39], targeting a 32-bit Arm Cortex-M4 micro-controller, confirm the findings in [44] on the suitability of said PQC candidates. These results show that there are PQC schemes that can be used on embedded platforms and hence are promising candidates for the integration into EV charging protocols.

3 System and Attacker Model

Our system model assumes the e-mobility architecture with the entities and communication relations shown in Fig. 1. We require that EVs supporting QuantumCharge are equipped with an HSM supporting the required PQC algorithms (cf. the analysis in [67]) which can be used to locally generate keys in the EV (similar to [26, 28]). Legacy EVs do not support QuantumCharge. Similarly, CPs and backend systems may or may not support PQC and QuantumCharge. Thus, we have several use cases ranging from all entities supporting QuantumCharge to no one supporting QuantumCharge.

The security of the PnC communication is of high importance as a successful attack can cause financial damages and may even harm power grid stability (cf. [1, 40, 69]). Thus, there are high incentives for attackers to compromise the security of the charging process [8]. In our attacker model, we distinguish attackers based on their area of influence. We consider the following four kinds of attackers with access to quantum computers. Moreover, we consider that cryptographic algorithms might become insecure during the lifecycle of vehicles and charging infrastructure, requiring cryptographic agility to be in place.

Compromised EV. We consider that the attacker has complete control over the charging vehicle. This encompasses remote attackers compromising the car as well as physical attacks. For example, the attacker could be the car owner or someone with temporary access in a car-sharing scenario. Moreover, car owners might be incentivized to manipulate the billing information to conduct charging fraud. Such an attacker might replace components of the car or temper with the firmware of ECUs, but does not have the capabilities to perform physical attacks on HSMs.

Compromised CP. We consider an attacker with complete control over the CP. As the billing information is generated by the CP, manipulating this data is difficult to prevent in case of a compromised CP. However, the attacker might

⁴ Arm Cortex-A53 platforms are increasingly used in more powerful automotive Electronic Control Units (ECUs), e.g., see Renesas product range [57].

try to manipulate, replay, or forge any data that a CP receives/forwards (e.g., metering confirmations generated by vehicles⁵).

Compromised Network. A network attacker has complete control over a network node between the CP and backend systems or between the CP and the EV. For example, a malicious device could have been installed within the charging cable or plug or directly behind the CP, allowing for manipulation of the sent and received communication. However, the attacker has no control over EV, CP, or backend components. As we aim for backward compatibility, downgrade attacks have to be considered with regard to this attacker (cf. Sect. 6).

Leaked Backend Data. An attacker might be able to extract information from backend systems. We assume that top-level secret keys, such as certificate signing keys, are sufficiently protected, but individual EV keys might not. Thus, private keys of EVs should not be stored in the backend.

4 Requirements

We define the requirements for security (*RS*) and feasibility (*RF*) as follows:

- RS*₁ *Secure key storage:* Private keys must be stored in a protected and secure environment to prevent their leakage.
- RS*₂ *Secure cryptographic operations:* Cryptographic algorithms must be executed within a secure, tamper-resistant, and separated environment.
- RS*₃ *Key usage authorization:* Key usage must be limited to authenticated and trusted software to prevent unauthorized access.
- RS*₄ *Secure key provisioning:* Private keys must be generated in a secure environment and must never leave it.
- RS*₅ *PQC support:* Cryptographic algorithms used for security-critical functions must resist attacks by quantum computers.
- RS*₆ *Crypto-agility:* The system must provide a mechanism to update or replace outdated or broken cryptographic algorithms securely.
- RS*₇ *Secure Credential Installation:* Bilateral authentication of the data sent between CPS and EV for credential installation must be guaranteed.
- RS*₈ *Secure Charge Authorization:* The authenticity of charge authorization requests received by the CP must be guaranteed.
- RS*₉ *Charge Data Authenticity:* The authenticity of received charge data as attested by the EV must be guaranteed towards the eMSP.
- RF*₁ *Minimal overhead.* The system should keep extra communication and computational overhead within the constraints of existing standards.
- RF*₂ *Easy integration.* The system should not alter the message flow of existing protocols and only introduce minor changes to message content.
- RF*₃ *Continued operation.* The system should offer backward compatibility with regard to actors that do not support PQC.

⁵ While ISO 15118 allows an EV to confirm meter values via a signature (called meter receipt), using this signature for billing purposes is subject to local regulation [36].

5 Security Concept

In this section, we describe our analysis of cryptographic algorithms which need to be replaced with PQC algorithms, the general approach of QuantumCharge, and detail the integration of QuantumCharge into the PnC architecture.

5.1 Analysis of PnC Cryptographic Algorithms

We focus our analysis of the specified cryptographic algorithms on both editions of ISO 15118 and relevant processes as described in Sect. 2.1. The application-specific protocols of ISO 15118 (discussed in the following) operate inside a TLS channel between the EV and the CP as well as the CP and the backend. Since there already exists exhaustive work on the migration of TLS to PQC (e.g., [16, 18, 20, 54, 59, 62]) and prototype PQC-TLS libraries (e.g., [63]), we consider TLS out of scope for this paper and assume a post-quantum secure underlying TLS channel in the following. Instead, we focus on application-layer security mechanisms, which need to be post-quantum secure independently of the individual TLS channels (e.g., credential installation between EV and the backend).

PKI and Signatures. ISO 15118 defines a (rather complex) Public Key Infrastructure (PKI) as the basis for credential installation and charge authorization. Public/private key pairs are used in the OEM provisioning and contract credentials. In addition, eMSPs, CPSs, and CPs are also equipped with public/private key pairs and corresponding certificates. All certificates and key pairs use Elliptic Curve Cryptography (ECC). Edition 1 uses the `secp256r1` curve and ECDSA with SHA-256 as the hash function. Edition 2 uses `secp521r1` and ECDSA with SHA-512 or `Curve448` and EdDSA with SHAKE256. The key pairs are used in the TLS handshake and on the application layer for signature generation. Any ECC keys and signature algorithms must be replaced with PQC alternatives.

The current NIST PQC API for signature algorithms assumes that complete messages and not message digests are signed. For simpler integration with XML signatures, we nevertheless keep the current approach of signing message digests also when using PQC algorithms, which introduces minor overhead but is directly interoperable with the existing protocols. Notably, while the XML signature specification includes a list of supported algorithms, it explicitly supports extensibility via application defined (in our case PQC) algorithms [10].

Key Establishment and Symmetric Encryption. The contract credentials installation process as standardized in ISO 15118 is based on an ephemeral-static ECDH key agreement for establishing a symmetric key (cf. Footnote 2) with the above-mentioned ECC curves for the respective editions. The hash function for the Key Derivation Function (KDF) is SHA-256 in Edition 1 and SHA-512 in Edition 2. The symmetric key is then used to encrypt the private contract key with AES-CBC-128 in Edition 1 and AES-GCM-256 in Edition 2. This classical ISO 15118 contract credentials installation process would require a suitable replacement for Diffie-Hellmann. For example, one could change the protocol to

use PQC algorithms instead of ECC and a PQC Key Encapsulation Mechanism (KEM) for key establishment. The key length for AES should be 256 bits and the length of the hash functions should be at least 512 bits. However, since we additionally integrate the approach from [26, 28] in QuantumCharge and generate keys securely in the EV's HSM (cf. Sect. 5.2), we do not require (PQC) key establishment and symmetric encryption in QuantumCharge anymore (except for the underlying TLS channels).

5.2 General Approach

QuantumCharge's general approach is to update ISO 15118 (and accordingly other PnC standards) to support PQC with minimal protocol changes while providing a high level of security. This is intended to enable easy adoption by the industry. QuantumCharge has five central parts as described in the following.

First, QuantumCharge integrates PQC algorithms based on our analysis described in Sect. 5.1. For signatures in TLS and at the application layer, we propose to use the PQC algorithms selected by NIST for standardization [49], i.e., the two lattice algorithms Dilithium [7] and FALCON [24] and the hash-based algorithm SPHINCS+ [6]. The idea is to provide a proof of concept with NIST-approved algorithms of different families to be able to switch between algorithms in case of compromise. On the application layer, we only need signatures since the encryption of private contact keys for transfer is replaced by a secure key generation in the EV's HSM. For TLS, we propose to replace Diffie-Hellmann with the KEM Kyber [14] (which is currently the only KEM selected by NIST), SHA-512 as hash function, and AES-GCM-256 for encryption. Dilithium, FALCON, and SPHINCS+ as well as Kyber are currently the only schemes that are being standardized by NIST; more schemes will likely follow in the future and then will be relevant in this context as well.

Second, QuantumCharge integrates a migration concept to address the issue of existing legacy PnC entities that cannot be updated to support PQC. Hereby we assume that only EVs and CPs are affected, but all backend systems can be updated. In addition to the PQC algorithms, the classical ECC algorithms, as standardized in both editions of ISO 15118 remain supported. At the beginning of an ISO 15118 charging session, EV and CP agree on the protocol version to be used, i.e., conventional ECC-based ISO 15118 or QuantumCharge. This protocol negotiation allows legacy ECC algorithms to remain supported. Note that negotiation may fail if EV or CP are configured to only accept PQC-secured connections.

Third, QuantumCharge provides crypto-agility by supporting multiple PQC algorithms and extending the ISO 15118 process with an algorithm negotiation (which is out of scope in the standard, cf. [V2G20-2320] in [37]). Algorithm negotiation is only performed if QuantumCharge was selected for this session and is extensible with any arbitrary (PQC) algorithms, e.g., in case the existing algorithms are broken or new more secure/efficient ones are developed. Again, negotiation may fail if EV and CP cannot agree on common algorithms.

Fourth, QuantumCharge generates all keys of the EV securely in an HSM in the EV. Keys for the EV are no longer generated in backend systems. Following the TPM-approach from [26, 28] we assume that a generic HSM with PQC support is used. The HSM is used for secure key generation, secure storage of (private) keys, and for secure execution of cryptographic operations. Further, the HSM provides additional features for restricting the usage of keys to a trustworthy system state (e.g., based on secure/authenticated boot).

Fifth, QuantumCharge includes changes to the credential installation and PnC authorization processes of ISO 15118 to allow for formal security proofs. Specifically, we enable the verification of strong formal authentication properties to meet the respective security requirements from Sect. 4.

5.3 Extending ISO 15118 with QuantumCharge

To enable the use of QuantumCharge, it must be supported by all involved entities (cf. Fig. 1). However, it is still possible for QuantumCharge-enabled EVs to use legacy CPs with classic ECC according to the original ISO 15118 standards. For security reasons, though, EVs can also be configured not to charge at such CPs. In the following, we describe QuantumCharge in more detail by describing the components and the integration into the ISO 15118 Edition 1 protocol flow, as Edition 1 is still the prominent edition of the protocol today. However, as the adoption of Edition 2 is expected to increase, we already explicitly consider the integration of QuantumCharge into ISO 15118-20 at the end of this section. The descriptions assume an error-free process. In the case of an error, the respective process may be aborted (e.g., as described above, if EV and CP fail to agree on an algorithm).

EV Manufacturing. In the EV production, an HSM supporting ECC and PQC algorithms is installed. We assume the HSM ensures that keys are generated in the HSM (and not imported) and that private key export is not allowed.

During customization, multiple key pairs $\{PC_{pk}^{a_i}, PC_{sk}^{a_i}\}$ are generated by the HSM and the OEM creates the corresponding *OEM provisioning certificates* $PC_{Cert}^{a_i}$, where a_i denotes the respective cryptographic algorithm of the set of supported algorithms $\{a_1, \dots, a_n\}$. In addition to the classic ECC algorithms EC-/EdDSA (for use with legacy systems), we support the PQC algorithms as mentioned in Sect. 5.2 with their different parameter sets. Key usage can be restricted to specific policies, e.g., access is only possible after a secure boot. We omit details on this aspect and refer to the TPM-based approach in [26, 28].

Contract Credential Usage. Figure 2 shows the PnC authorization process using the contract credential. First, the TLS channel between EV and CP is established (not shown)⁶. When a communication session is initiated by an EV, a

⁶ Since PQC for TLS is addressed in other work (see Sect. 1), we omit the details on TLS in the following and discuss only the application layer of ISO 15118. In our current design, the choice of algorithms is done independently for TLS and application layer but could also easily be coordinated.

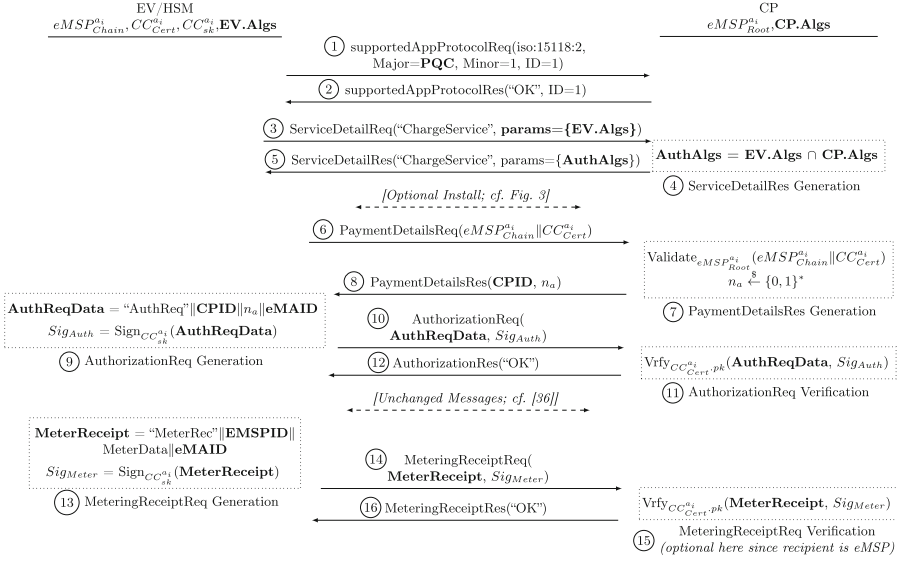


Fig. 2. Usage procedures of contract credentials (changes in bold).

handshake is performed to negotiate the used application protocol and version. More specifically, the standard defines a major and a minor version number. The major number is used for the protocol version of either Edition 1 or 2. The minor number can indicate minor changes of the protocol (e.g., additional data elements). The EV sends a prioritized list of supported application layer protocols/versions to the CP, which responds with its selection.

To enable QuantumCharge and support backwards compatibility, we extend this initial application protocol/version negotiation. Notably, the minor version number cannot be used to indicate the use of PQC since with ISO 15118-20 a divergence in minor numbers is not indicated to the EV by the CP (if the major number matches and the minor number is lower, the CP simply responds with an *OK*; cf. [37] [V2G20-170]). To detect that the backwards compatibility mode is needed, however, the EV would require this indication. Hence, we show QuantumCharge support via the definition of new major numbers such that there are now four rather than two valid flags for the major number (for Edition 1 or 2, each with or without PQC). The major and minor numbers are used in the prioritized list sent by the EV (Step 1 in Fig. 2). The CP selects the protocol version and replies with an *OK* (Step 2).

If both sides support QuantumCharge, an additional handshake is performed to negotiate the signature algorithm using modified service detail messages (for the “ChargeService”; Steps 3-5). The EV sends a prioritized list of supported algorithms to the CP. The CP responds with its selection. At this point, the EV may start a contract credential installation as detailed later in this section.

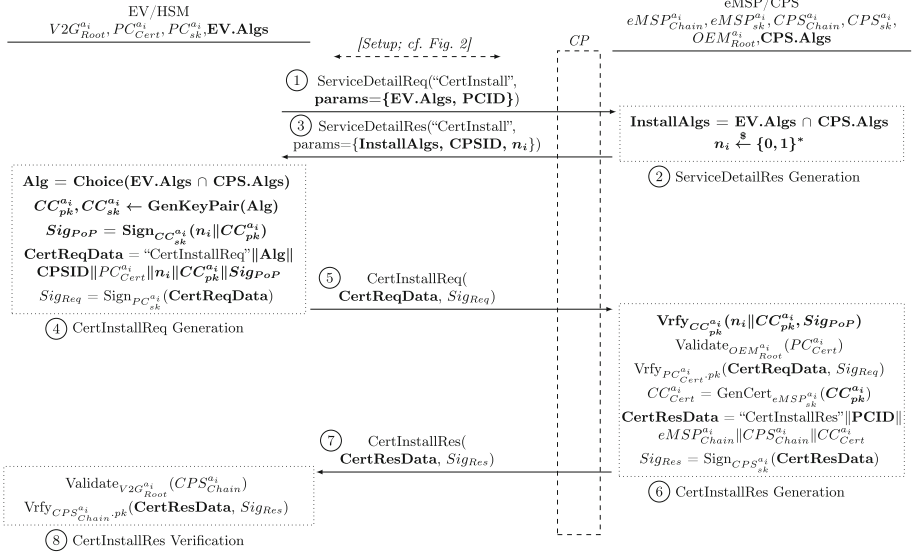


Fig. 3. Installation procedure of a contract credential (changes in bold).

If the EV possesses contract credentials using the chosen algorithm (from Step 5), it can start the PnC authorization process by sending its corresponding contract certificate chain to the CP (Step 6). The CP validates the chain⁷ based on a locally installed eMSP root and responds with its unique ID and a fresh nonce n_a (Step 7-8). Afterwards, the EV uses its HSM to sign⁸ its authorization request data (including eMAID and the CP's ID to identify the sender/recipient of this signature, which is essential for the verification of formal authentication properties) and sends the signed data to the CP (Steps 9-10). The CP verifies the signature using the public key from the previously (in Step 6) received $CC_{Cert}^{a_i}$ and if the signature is valid, the EV is cleared to charge (Steps 11-12).

Similar to authorization request signing, an EV can use its PQC contract credentials to sign meter receipts (Step 13). The signed data again includes the eMAID and an ID of the recipient (i.e., the eMSP for, e.g., billing purposes) and is sent to the CP, which may verify the signature using the EV's public contract key (Steps 14-16). The meter receipt is later forwarded to the eMSP, along with

⁷ Note regarding notation: in Fig. 2 and Fig. 3 we use $\text{Validate}_{Cert_{Root}}(Cert_{Chain})$ to denote the validation of the certificate chain $Cert_{Chain}$ based on the root certificate $Cert_{Root}$ using the default certificate chain validation algorithm from [19].

⁸ Note regarding notation: in Fig. 2 and Fig. 3 we use $Sig = \text{Sign}_{X_{sk}}(Data)$ to denote the generation of a cryptographic signature Sig over $Data$ using the private key X_{sk} . We use $\text{Vrfy}_{X_{pk}}(Data, Sig)$ to denote the corresponding signature verification with the public key X_{pk} . Additionally, we use $X_{Cert.pk}$ to denote the extraction of a public key from a certificate.

other billing-relevant data, and the eMSP verifies the signature (not shown in Fig. 2).

Contract Credential Installation/Update. After the application protocol and signature algorithm negotiations between EV and CP are completed, the EV may choose to install new contract credentials (e.g., if it does not possess valid credential with the chosen algorithm). For this, the HSM generates the key pairs $\{CC_{pk}^{a_i}, CC_{sk}^{a_i}\}$ and the EV requests the eMSP to issue the corresponding *contract certificates* $CC_{Cert}^{a_i}$ for the public keys $CC_{pk}^{a_i}$.

Figure 3 shows the process for installing contract credentials using the modified *CertInstallReq* message. The process again starts with an algorithm negotiation using modified service detail messages (for the “CertInstall”; Steps 1-3). The EV sends (via the CP) a prioritized list of supported algorithms to the CPS, which acts as a middleman between EVs and eMSPs in the installation process (Fig. 3 combines CPS and eMSP for simplicity). The CPS responds with a selection of accepted algorithms and a fresh nonce n_i . The EV uses its HSM to generate the contract credential key pair $\{CC_{pk}^{a_i}, CC_{sk}^{a_i}\}$, create a Proof-of-Possession (PoP) signature over n_i and $CC_{pk}^{a_i}$ with $CC_{sk}^{a_i}$, and sign the certificate request data using $PC_{sk}^{a_i}$ (Step 4). The credential update process is similar except that an old $CC_{sk}^{a_i}$ is used for signing instead of $PC_{sk}^{a_i}$ (not shown).

The signed certificate request data, which includes the selected algorithm, an ID of the recipient, the $PC_{Cert}^{a_i}$, n_i , $CC_{pk}^{a_i}$, and the PoP signature, is sent (via the CP) to the CPS (Step 5). The CPS verifies the PoP signature based on $CC_{pk}^{a_i}$, validates the provisioning certificate $PC_{Cert}^{a_i}$ based on a locally installed OEM root, verifies the certificate request data signature based on $PC_{Cert}^{a_i}$, and forwards the request to the eMSP (Step 6). Afterwards, the eMSP responds with a contract certificate $CC_{Cert}^{a_i}$ for $CC_{pk}^{a_i}$ based on which the CPS can build and sign the certificate response data (Step 6). The signed response data is sent to the EV, which verifies the CPS signature based on a locally installed root and stores the contract credential for later use (Steps 7-8).

Integration into ISO 15118-20. In case Edition 2 of ISO 15118 is supported, minor changes to the different processes are required. Firstly, the EV manufacturing process now also requires that the EV’s key pairs and corresponding vehicle certificates are generated for usage in the TLS handshake. Key pairs are again generated in the EV’s HSM and the OEM reads out the public keys and generates the certificates.

Secondly, compared to Edition 1, ISO 15118-20 slightly changes the contract credential usage message flow. The most important change is that *ServiceDetail* messages are sent after authorization and thus cannot be used for algorithm negotiation. Instead, we propose the use of the new *Authorization-Setup* messages for this purpose, whereby the EV’s request (usually empty) contains *EV.Algs* and the CP’s response (usually only n_a and some information about the offered services) additionally contains *AuthAlgs* and *CPID*. Furthermore, ISO 15118-20 defines no *PaymentDetails* messages as the CP’s nonce n_a is included in the *AuthorizationSetupRes* message and the EV’s contract certificate

chain is included in its *AuthorizationReq*. Respectively, QuantumCharge would also include the EV’s contract certificate chain in its *AuthorizationReq*.

Finally, the contract credential installation process requires minor adjustments. In ISO 15118-20, *ServiceDetail* messages are sent after authorization and, thus, after credential installation. Additionally, the installation of multiple credentials from different eMSPs is supported, whereby multiple *CertInstall* message pairs are exchanged. We thus propose the use of *CertInstall* messages for algorithm negotiation, whereby the first *CertInstall* message pair contains the data of the *ServiceDetail* message pair in Fig. 3. Following *CertInstall* message pairs contain the proposed installation data with no further changes needed.

6 Formal Security Verification

In our formal analysis, we focus on the changes made to the authorization protocol. Verifying the security of HSMs or backend infrastructure is outside the scope of this paper. Thus, we formally verify RS_6 to RS_9 below. For our other security requirements, please note that RS_1 , RS_2 , and RS_3 are addressed by the usage of an HSM with PQC support. Hereby, private keys are stored and used in a secure area of the HSM (addressing RS_1 and RS_2). In addition, we assume that the HSM enforces key usage authorization, e.g., using a policy that enables key access only after a secure boot process (addressing RS_3). We assume that keys stored within the backend, especially Certificate Authority (CA) keys, will be processed at least as securely, for example, using a server HSM and additional physical access control to the processing hardware. Moreover, RS_4 is enabled by our changes to the credential installation process, as private credential keys are no longer generated in the backend but in the EV’s HSM instead. Possible downgrade attacks should be addressed by timely disabling insecure algorithms by policies, especially the backwards compatibility with non-PQC algorithms should be disabled when quantum computer-based attacks become feasible for malicious actors. This will require close negotiation between the OEMs, eMSPs, and Charge Point Operators to determine the best security/operability trade-off for the specific ecosystem. We abstract from the used algorithms and assume they satisfy RS_5 .

We formally verify our protocol in the symbolic model, also called the Dolev-Yao model [22]. In this model, cryptographic primitives are represented by symbolic functions and assumed to be perfectly secure. Instead, the security of the composition of these primitives is analyzed. Usually, the attacker has complete control over the network (cf. the *Compromised Network* adversary in Sect. 3). However, restrictions of this control, for example, by assuming secure or authenticated channels, are possible. In addition, we allow the presence of multiple dishonest parties, i.e., the attacker can corrupt multiple parties that are not directly involved in the transaction under proof. For example, if Alice sends a message to Bob, this message has to remain secure even if there is a compromised Charlie that also sends messages to Alice and Bob.

We use the Tamarin prover [47] to verify the security of our model, a state-of-the-art tool for automated symbolic protocol verification. In Tamarin, a model

is specified as a set of rules that define a protocol’s communication and data-processing steps. Security properties that are required to hold over all possible execution traces of the model are specified in first-order logic and called lemmas. Tamarin starts with a state where the property has been violated and performs a backward search over the possible rule executions to determine whether there is a valid path that leads to this state. If there is, the property does not hold, and Tamarin has found a counterexample. If there is no possible execution path that can lead to a violation of the property, this proves that the property holds.

Our complete Tamarin model files for QuantumCharge are provided in an online repository.⁹ The files contain lemmas to verify the desired security properties and additional lemmas that verify the correctness of the Tamarin specification of our model. The repository also includes instructions on how to run the model and reproduce and verify the results of our formal analysis and details on the verification times.

We model the authentication requirements from RS_7 to RS_9 with the notion of *injective agreement*. Injective agreement is a well-established and strong authentication property originally defined by Lowe [42] and commonly used in current research (e.g. [11, 29, 41, 68]). It encompasses verification of the identity of the communication partners, the integrity of the exchanged messages, and protection against replay attacks.

Definition 1 (Injective agreement). *A protocol guarantees injective agreement to an honest initiator A with an honest responder B on a set of data items \mathbf{ds} if, whenever A , acting as initiator, completes a run with the protocol, apparently with responder B , then B has previously been running the protocol, apparently with A , and B was acting as a responder in this run. Moreover, each run of A corresponds to a unique run of B and both agents agree on the values of the variables in \mathbf{ds} [42].*

To encompass crypto-agility (RS_6), cryptographic operations in our model always reference an algorithm under which they are computed. Algorithms can become insecure at any time during the protocol run, which is modeled by a rule that exposes the secret key of a party, allowing the attacker to forge signatures freely. Moreover, parties can revoke support for algorithms they consider insecure, aborting all pending operations using this algorithm. Ideally, we would like to show that *injective agreement* holds for all transactions in our model, given that each insecure algorithm has been revoked before becoming insecure by at least one of the involved parties before being broken by the attacker. However, this property is unrealistically strong since an attack would always be possible against an entity using insecure algorithms. For example, the EV could send a message to the CPS, then the used algorithm becomes insecure and is revoked by the EV but not by the CPS. The message would then be accepted by the CPS regardless. We argue that the actual risk of such an attack is limited, as its possible timeframe is small. However, we can only show a slightly weaker property, requiring that *injective agreement* holds for parties that revoked the

⁹ <https://code.fbi.h-da.de/seacop/quantumcharge-source>

algorithm before it became insecure, but not necessarily for their communication partner.

Definition 2 (Injective agreement under insecure algorithms). *Honest A with honest B injectively agree (cf. Def. 1) on \mathbf{ds} and all cryptographic algorithms used in this protocol run, either directly by A and B or within the setup of credentials they use during this protocol run, either remain secure during the entire protocol run or are revoked by A prior to becoming insecure.*

Using Def. 2, we formally verify that QuantumCharge provides RS_7 , RS_8 , and RS_9 with the Tamarin prover, under consideration of RS_6 by modeling crypto-agility as previously discussed. For this, all communication of CPs is assumed to be under the control of the adversary (cf. *Compromised CP* in Sect. 3). Entities that are directly involved in a specific protocol run (EV, CPS, or eMSP) are assumed to be *honest*, i.e., we assume that their cryptographic credentials have not been revealed to the adversary. This assumption is reasonable due to the use of HSMs (cf. RS_1 , RS_1 , and RS_3). However, in order to keep the needed assumptions as weak as possible, other entities of the same types that are not directly involved in the protocol run can be compromised (cf. *Compromised EV* and *Leaked Backend Data* in Sect. 3). Thus, our model can verify that, even if the key storage/usage of some entities is subject to an attack (e.g., due to an implementation error in some HSMs), the security of all other entities remains intact.

The execution of Tamarin with our model⁹ shows that QuantumCharge satisfies the security properties, as no counterexamples are found. As the proof generation is fully automated, the following description summarizes the verified properties instead of going into proof details.

Secure Credential Installation. For the security of the credential installation process (RS_7), we consider a compromised CP and network. Note that, as we always allow for the presence of additional dishonest parties, the presence of compromised EVs trying to impersonate a different EV during the process is also considered. Using our Tamarin model, we verify that *injective agreement under insecure algorithms* holds for credential installation transactions in our protocol between EV and CPS in both directions. That is, the CPS (as initiator) *injectively agrees* with the EV (as responder) on the certificate installation request (Step 5 in Fig. 3 and the EV (as initiator) *injectively agrees* with the CPS (as responder) on the certificate installation response message (Step 7 in Fig. 3). As an example, we look at the first of these properties in detail in Appendix A.1.

Secure PnC Authorization Process. We verify that the CP and the EV *injectively agree under insecure algorithms* on authorization requests from the EV. That is, the Tamarin analysis shows that QuantumCharge provides strong security for charge authorizations (RS_8). Note that the algorithms used during the setup of the used credentials are assumed to have remained secure during their setup.

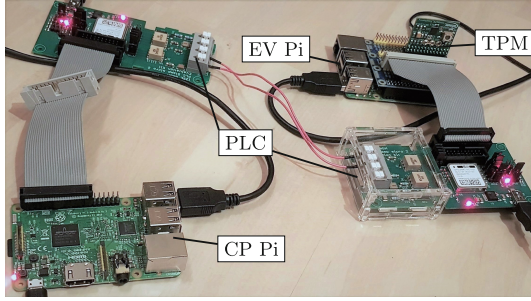


Fig. 4. Proof-of-Concept Setup

Charge Data Authenticity. As charge measurements originate from the CP’s meter, it is difficult to prevent the CP from providing false meter data to the eMSP for billing. In ISO 15118, this is addressed by optional metering receipts. The EV may compare the CP’s meter data with its own measurements before signing receipts. Our analysis shows that for metering receipts verified by the eMSP, *injectiv agreement under insecure algorithms* on the included data (RS_9) is satisfied between the eMSP and the EV. Thus, the data cannot be manipulated by the CP or a network attacker.

7 Implementation and Practical Feasibility Evaluation

We implemented QuantumCharge as a proof-of-concept. The setup is shown in Fig. 4 and the code is provided online.⁹ The EV and CP are implemented on Raspberry Pi 3 Model B+ boards and connected via PLC stamp micro 2 EVBs to emulate Power Line Communication as per ISO 15118. The Pi boards have an Arm-based Quad Core CPU at 1.4 GHz and 1 GB RAM. While there is variance between manufacturers, this setup can nonetheless be argued to be representative since: (i) w.r.t. the CP’s side, commercial controllers with comparable performance exist, such as Vector’s vSECC [66] (using an Arm-based Quad Core CPU at 1 GHz and 2 GB RAM) and (ii) w.r.t. the EV’s side, an option that is discussed in the industry is the offloading of cryptographic functions for PnC to one of the more capable vehicle ECUs such as the infotainment/head unit, which is usually in the Arm Cortex-A performance class [4] (e.g., [64] with a dual Arm Cortex-A15 at 1.2 GHz).

Our ISO 15118 implementation uses RISE V2G [65] in Java, with the changes proposed in Sect. 5. For the PQC algorithms, we use the OQS implementations in C, called from Java via JNI using the provided bindings. Certificates and keys for the PKI are generated using OQS’ PQC OpenSSL prototype. Functions of the e-mobility backend (CPS/eMSP) are implemented on the CP-Pi for simplicity.

For each PQC algorithm, all working parameter options supported by OQS are included in our QuantumCharge implementation and evaluation. In the following, we only list the results of one parameter set per algorithm; the full results

Table 1. Communication overhead (in byte).

Message	Algorithm					
	ecdsa256	ecdsa521	eddsa448	dilithium2	falcon512	sphincssha256 128frobust
PaymentDetailsReq Total	1472	1880	1489	12262	5679	52371
<i>eMSPChain</i> <i>CCCert</i>	1435	1843	1452	12225	5642	52331
AuthorizationReq Total	383	452	426	2735	974	17417
<i>SigAuth</i>	71	139	114	2420	660	17088
CertInstallReq Total	1150	1536	1229	11868	5006	57781
<i>CCpk</i>	91	158	69	1312	897	32
<i>SigPoP</i>	72	139	114	2420	660	17088
<i>PCCert</i>	478	614	489	4080	1885	17449
<i>SigReq</i>	71	138	114	2420	659	17088
CertInstallRes Total	3313	4196	3406	27261	12327	122162
<i>eMSPChain</i> <i>CCCert</i>	1435	1843	1452	12225	5642	52331
<i>CPSChain</i>	1422	1828	1455	12228	5638	52335
<i>SigReq</i>	71	139	114	2420	660	17088

can be found online.⁹ Specifically, we include PQC parameter sets for NIST security level 1, which is comparable to the current security of ISO 15118. The only exception is Dilithium, where we use security level 2, since Dilithium does not provide level 1 parameters.

We use the proof-of-concept implementation to measure the overhead of QuantumCharge in comparison to the default ISO 15118 algorithms. W.r.t. communication overhead, the most significant changes come from the inclusion of PQC public keys and signatures in various messages. Table 1 provides an overview of the resulting message sizes and the most significant element sizes.¹⁰

It shows only minor differences between the non-PQC algorithms. The variance between the PQC algorithms, however, is relatively high, with FALCON offering the lowest overhead (closest to the non-PQC algorithms) due to its relatively small public keys and signatures. Since ISO 15118 messages use a 4-byte length field, their size is limited to 4,294,967,295 byte, which is not violated by any of the evaluated algorithms.

The overhead for *MeteringReceiptReq* messages is comparable to that of the *AuthorizationReq* since both cases involve the same cryptographic actions (a signature with the private contract key) and is thus not explicitly listed. Additional minor overhead (not detailed in Table 1 but part of the totals) is caused by: (i) The inclusion of algorithm IDs in different messages (e.g., a 2-byte ID per algorithm similar to TLS). (ii) The addition of the CPID in *PaymentDetailsRes* and *AuthorizationReq* messages (39-255 characters [37]). (iii) The addition of

¹⁰ Message size totals are slightly larger than the sum of element sizes since totals represent the size of EXI-encoded XML messages (as sent between EV and CP) and since element sizes that are independent of the algorithm are omitted for simplicity.

Table 2. Storage overhead (in byte).

Message	Algorithm					
	ecdsa256	ecdsa521	eddsa448	dilithium2	falcon512	sphincssha256 128frobust
EV Cryptographic Data Total	3558	4709	3538	36269	17569	122342
V2G Root Certificate	472	608	483	4074	1882	17443
Provisioning Cert. Chain	1429	1835	1461	12234	5649	52341
Provisioning Key Pair	121	223	73	3870	2202	115
Contract Certificate Chain	1415	1820	1448	12221	5634	52328
Contract Key Pair	121	223	73	3870	2202	115
CP Cryptographic Data Total	2480	3259	2486	24238	11589	87328
V2G Root Certificate	472	608	483	4074	1882	17443
eMSP Root Certificate	468	602	479	4070	1876	17439
CP TLS Certificate Chain	1419	1826	1451	12224	5629	52331
CP TLS Key Pair	121	223	73	3870	2202	115

the CPSID in specific *ServiceDetailRes* and *CertInstallReq* message (e.g., 2 char country code plus 3 char operator ID similar to eMSP IDs [37]).

The storage overhead of QuantumCharge for EV and CP is shown in Table 2. The reported sizes for all data elements are based on their DER encoded [34] format. All certificate chains use two Sub-CAs, i.e., the chains are the maximum allowed length of ISO 15118. Regarding storage overhead, our results again show only minor differences between the non-PQC algorithms and a high variance between the PQC ones with relations comparable to that of the communication overhead. Notably, a potential compatibility issue of QuantumCharge is created since ISO 15118, independently of its XML message definitions, limits the maximum size of DER-encoded certificates to 800 bytes. In ISO 15118-20, this limit is increased to 1600 byte. Since none of the evaluated PQC algorithms can meet this limit, any kind of PQC-ready ISO 15118 would need a further increase and thus affect the storage requirements of involved systems.

W.r.t. computational overhead, the most significant changes of QuantumCharge are PQC key pair generation, signing, and signature verification. Timing measurements are repeated 100 times using Java’s *System.nanoTime()* and Table 3 shows the resulting averages. We see that most PQC algorithms are suited for the use case. Notably, Dilithium and FALCON are sometimes even faster than the standard algorithms. However, signature generation with the SPHINCS+ algorithm starting from security level 3 and using the small s-parameter sets (not shown) was too slow to meet ISO 15118’s limits (fast f-parameter sets showed no issues). Specifically, *CertInstallReq* generation was with 49s over the relevant ISO 15118 limit of 40s.

The performance evaluation shows that QuantumCharge mostly maintains compatibility with the current ISO 15118 limits and the only general (for all algorithms) issue arises from the larger certificate sizes. Since the respective limit, however, was already increased with ISO 15118-20, we argue that a further increase for PQC support is reasonable. The evaluation shows that for

Table 3. Computational overhead (in ms rounded to four significant figures).

Message	Algorithm					
	ecdsa256	ecdsa521	eddsa448	dilithium2	falcon512	sphincsha256 128frobust
PaymentDetailsReq Handling (CP)	102.4	325.2	82.67	39.28	20.1	279.5
Validate Certificate Path	99.59	322.2	79.63	35.80	17.06	272.9
AuthorizationReq Generation (EV)	42.39	85.05	51.45	30.44	60.74	1,157
Generate XML Signature	31.23	73.50	39.61	21.39	52.11	1,147
AuthorizationReq Handling (CP)	52.97	129.2	45.80	27.75	22.35	100.7
Verify XML Signature	46.27	122.4	39.24	21.39	16.29	93.59
CertInstallReq Generation (EV)	2,508	2,767	1,113	507.6	693.3	2,192
Generate Key Pair	2,372	2,552	807.1	42.04	230.2	71.18
Generate PoP Signature	29.73	72.66	40.90	12.33	38.46	824.3
Generate XML Signature	41.53	78.44	50.32	29.46	51.41	786.0
CertInstallReq Handling (CP)	190.2	407.6	188.9	235.2	225.0	3,582
Verify PoP Signature	54.81	87.75	27.47	7.396	3.078	48.62
Verify XML Signature	29.11	74.08	30.37	18.07	15.84	64.76
Validate Provisioning Certificate	27.11	64.95	26.66	14.19	8.631	64.86
CertInstallRes Generation	69.94	172.9	94.59	175.4	183.3	3,373
Generate Contract Certificate	16.95	52.92	17.50	19.74	55.89	1,299
Generate XML Signature	18.24	70.70	18.07	16.46	51.59	1,340
CertInstallRes Handling (EV)	208.6	456.5	245.1	143.0	114.2	536.8
Verify XML Signature	60.06	119.9	71.90	41.56	35.54	117.0
Validate Certificate Path	102.7	288.6	123.1	58.57	38.33	316.8

SPHINCS+, it is preferable to use the f-parameter sets instead of the s-parameter sets. Since a variety of PQC algorithms (for all security levels) are compatible with existing time limits, we consider RF_1 met. Further, as QuantumCharge is compatible with ISO 15118’s current roles and data flows, we argue that RF_2 is met. Moreover, as our extension supports backwards compatibility, RF_3 is met.

8 Conclusion

In this paper, we propose QuantumCharge, a PQC extension for ISO 15118. We analyze the PnC standards and protocols and show where PQC algorithms are required. QuantumCharge includes concepts for migration, crypto-agility, verifiable security, and the use of PQC-enabled HSMs. As baseline, QuantumCharge implements Dilithium, FALCON, and SPHINCS+ and can be extended easily with other PQC algorithms. With our prototype, we analyze the introduced communication and computational overhead. Our results show that all PQC algorithms require increasing the defined maximum certificate size in both editions of ISO 15118 and that all algorithms can meet the defined timing requirements. Hence, all PQC signature algorithms Dilithium, FALCON, and SPHINCS+ (using the fast f-parameter sets) selected by NIST for standardization can be used in QuantumCharge. Our formal analysis using Tamarin shows the security of our protocol changes. QuantumCharge provides strong security guarantees, user-friendly performance, and high compatibility to ISO 15118 as well as legacy systems, thus paving the way for PQC-secure charging.

Acknowledgements. This research work has been partly funded by the German Federal Ministry of Education and Research and the Hessian State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cyber-Security ATHENE and through the Taiwan Ministry of Science and Technology grant 109-2222-E-001-001-MY3.

A Appendix

A.1 Tamarin Lemma for RS_7

In Listing A.1, we give the Tamarin lemma for *injective agreement under secure algorithms* for the CPS. The `Commit_CPS_Install` event in Line 3 denotes that the CPS accepted a credential installation request by the client (identified by its PCID). The CPS sends accepted requests to the eMSP who generates the client's certificate, including the client's public key `cc_pub`, and sent back to the CPS. The CPS signs the response and sends it to the client (cf. Step 5 to 7 in Fig. 3). We require that for all such commit events, there is a corresponding event `Running_EV_Install`, denoting that the EV identified by PCID previously sent a certificate installation request (cf. Step 5 in Fig. 3) for the public contract credential key `cc_pub` (Line 4 to 6). In addition, there must not be an additional commit event for the same public key `cc_pub` by any certificate provisioning service (Lines 7-9). This ensures that the `Running_EV_Install` event corresponds to a unique `Commit_CPS_Install` event, i.e., that no replay attacks are possible. We only require this property to hold if all algorithms used by the involved parties either remain secure during the entire transaction or have been revoked by the CPS before completing the transaction (Lines 10-14). This includes the algorithm used for the provisioning certificate of the EV, as well as the signature testifying its validity, the algorithm used for the contract credentials (`cc_pub`), and the algorithm used by the CPS's certificate chain. Note that the CPS will abort the transaction if it uses revoked algorithms. We model the insecurity of an algorithm the same way as the corruption of an entity, that is, the private key of a credential set for this algorithm and entity is given to the attacker, which is denoted by a `KeyReveal` event.

```

1 lemma auth_install_inj_agreement_insec_algs_CPS :
2   " All CPS PCID cc_pub #i .
3     Commit_CPS_Install(CPS,PCID,cc_pub) @i
4     ==> ( (Ex #j .
5           Running_EV_Install(PCID,CPS,cc_pub) @j
6           & (#j<#i)
7           & not( Ex CPS2 PCID2 #i2 .
8                 Commit_CPS_Install(CPS2,PCID2,cc_pub) @i2
9                 & not(#i2=#i) ) )
10          | (Ex entity alg #kr .
11              KeyReveal(entity , alg) @kr
12              & Honest(entity , alg) @i
13              & not(Ex #kr2 . E.RevokedAlg(CPS, alg) @kr2
14                  & kr2<i) ) ) "

```

Listing A.1. Injective Agreement under Insecure Algorithm Lemma in Tamarin.

References

1. Acharya, S., Dvorkin, Y., Karri, R.: Public plug-in electric vehicles+ grid data: Is a new cyberattack vector viable? *IEEE Trans. Smart Grid* **11**(6), 5099–5113 (2020)
2. Alnahawi, N., Schmitt, N., Wiesmaier, A., Heinemann, A., Graßmeyer, T.: On the State of Crypto Agility. In: *Tagungsband zum 18. Deutschen IT-Sicherheitskongress*, vol. 18, pp. 103–126. German Federal Office for Information Security (BSI) (2022)
3. Alnahawi, N., et al.: On the state of post-quantum cryptography migration. In: *INFORMATIK 2021*, pp. 907–941. Gesellschaft für Informatik, Bonn (2021)
4. Arm: a starter’s guide to arm processing power in automotive (2018). <https://community.arm.com/arm-community-blogs/b/embedded-blog/posts/a-starters-guide-to-arm-processing-power-in-automotive>
5. Atkins, D.: *Requirements for Post-Quantum Cryptography on Embedded Devices in the IoT* (2021)
6. Aumasson, J.P., et al.: *SPHINCS+ Submission to the NIST post-quantum project, v.3* (2020)
7. Bai, S., et al.: *CRYSTALS-Dilithium - Algorithm Specifications and Supporting Documentation* (2021)
8. Bao, K., Valev, H., Wagner, M., Schmeck, H.: A threat analysis of the vehicle-to-grid charging protocol ISO 15118. *Comput. Sci.-Res. Develop.* **33**(1–2), 3–12 (2018)
9. Barker, W., Polk, W., Souppaya, M.: Getting ready for post-quantum cryptography: explore challenges associated with adoption and use of post-quantum cryptographic algorithms. Tech. rep., NIST Publications (2020). <https://doi.org/10.6028/NIST.CSWP.05262020-draft>
10. Bartel, M., Boyer, J., Fox, B., LaMacchia, B., Simon, E.: *Signature Syntax and Processing Version 1.1*. W3C recommendation, World Wide Web Consortium (W3C) (2013)
11. Basin, D., Sasse, R., Toro-Pozo, J.: Card brand mixup attack: bypassing the {PIN} in {non-Visa} cards by using them for visa transactions. In: *30th USENIX Security Symposium (USENIX Security 21)*, pp. 179–194 (2021)

12. Basu, K., Soni, D., Nabeel, M., Karri, R.: [Post-Quantum Cryptography - A Hardware Evaluation Study](#). Cryptology ePrint Archive, Report 2019/047 (2019)
13. Bindel, N., McCarthy, S., Rahbari, H., Twardokus, G.: [Suitability of 3rd Round Signature Candidates for Vehicle-to-Vehicle Communication – Extended Abstract](#). 3rd PQC standardization conference, NIST (2021)
14. Bos, J., et al.: [CRYSTALS – Kyber: A CCA-Secure Module-Lattice-Based KEM](#). In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 353–367 (2018)
15. Bova, F., Goldfarb, A., Melko, R.G.: [Commercial applications of quantum computing](#). EPJ Quantum Technol. **8**(1), 2(2021)
16. Bürstinghaus-Steinbach, K., Krauß, C., Niederhagen, R., Schneider, M.: [Post-Quantum TLS on Embedded Systems: Integrating and Evaluating Kyber and SPHINCS+ with Mbed TLS](#). In: ACM Asia Conference on Computer and Communications Security, pp. 841–852. ASIA CCS 2020, ACM (2020)
17. Campos, F., Meyer, M., Sanwald, S., Stöttinger, M., Wang, Y.: [cryptography for ECU security use cases](#). In: 17th escar Europe: embedded security in cars (conference proceedings). Ruhr-Universität Bochum (2019)
18. Chang, Y.A., Chen, M.S., Wu, J.S., Yang, B.Y.: [SSL/TLS for Embedded Systems](#). In: IEEE Conference on Service-Oriented Computing and Applications, pp. 266–270. IEEE (2014)
19. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, T.: Internet x.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 5280, RFC Editor (5 2008). <http://www.rfc-editor.org/rfc/rfc5280.txt>
20. Crockett, E., Paquin, C., Stebila, D.: [post-quantum and hybrid key exchange and authentication in TLS and SSH](#). Cryptology ePrint Archive, Report 2019/858 (2019)
21. Dang, V.B., Farahmand, F., Andrzejczak, M., Mohajerani, K., Nguyen, D.T., Gaj, K.: [Implementation and Benchmarking of Round 2 Candidates in the NIST Post-Quantum Cryptography Standardization Process Using Hardware and Software/Hardware Co-design Approaches](#). Cryptology ePrint Archive, Report 2020/795 (2020)
22. Dolev, D., Yao, A.: [On the security of public key protocols](#). IEEE Trans. Inf. Theory **29**(2), 198–208 (1983)
23. European Telecommunications Standards Institute (ETSI): Migration strategies and recommendations to quantum safe schemes. TR 103 619 V1.1.1 (2020)
24. Fouque, P.A., et al.: [Fast-Fourier Lattice-based Compact Signatures over NTRU Specification v1.2](#) (2020)
25. Fritzmam, T., Vith, J., Sepúlveda, J.: [Post-Quantum Security for Automotive Systems](#). In: Euromicro Conference on Digital System Design (DSD), pp. 570–576 (2020)
26. Fuchs, A., Kern, D., Krauß, C., Zhdanova, M.: [HIP: HSM-Based Identities for Plug-and-Charge](#). In: Conference on Availability, Reliability and Security. ARES 2020, ACM (2020)
27. Fuchs, A., Kern, D., Krauß, C., Zhdanova, M.: [TrustEV: Trustworthy Electric Vehicle Charging and Billing](#). In: ACM/SIGAPP Symposium on Applied Computing SAC 2020. ACM (2020)
28. Fuchs, A., Kern, D., Krauß, C., Zhdanova, M., Heddergott, R.: [HIP-20: Integration of Vehicle-HSM-Generated Credentials into Plug-and-Charge Infrastructure](#). In: Computer Science in Cars Symposium. CSCS 2020, ACM (2020)

29. Gazdag, S.L., Grundner-Culemann, S., Guggemos, T., Heider, T., Loebenberger, D.: A formal analysis of ikev2's post-quantum extension. In: Annual Computer Security Applications Conference, pp. 91–105. ACSAC 2021, Association for Computing Machinery, New York, NY, USA (2021). <https://doi.org/10.1145/3485832.3485885>
30. Gupta, D.S., Ray, S., Singh, T., Kumari, M.: [Post-quantum lightweight identity-based two-party authenticated key exchange protocol for Internet of Vehicles with probable security](#). Computer Communications **181**, 69–79 (2022)
31. Hülsing, A., Rijneveld, J., Schwabe, P.: ARMed SPHINCS. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 446–470. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_17
32. Hülsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: [XMSS: eXtended Merkle Signature Scheme](#). RFC 8391 (2018)
33. Hülsing, A., Ning, K.C., Schwabe, P., Weber, F., Zimmermann, P.R.: [Post-quantum WireGuard](#). In: IEEE Symposium on Security and Privacy, pp. 304–321 (2021)
34. International Telecommunication Union: Information technology-ASN.1 encoding rules: Specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER). ITU-T recommendation X.690 (2021)
35. ISO/IEC: Road vehicles - vehicle to grid communication interface - part 1: General information and use-case definition. ISO 15118-1:2013, ISO (2013)
36. ISO/IEC: Road vehicles - vehicle-to-grid communication interface - part 2: Network and application protocol requirements. ISO 15118-2:2014, ISO (2014)
37. ISO/IEC: Road vehicles - vehicle to grid communication interface - part 20: 2nd generation network and application protocol requirements. ISO 15118-20:2022, ISO (2022)
38. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: PQM4: Post-quantum crypto library for the ARM Cortex-M4. <https://github.com/mupq/pqm4>
39. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: [pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4](#). Cryptology ePrint Archive, Report 2019/844 (2019)
40. Kern, D., Krauß, C.: Analysis of e-mobility-based threats to power grid resilience. In: Proceedings of the 5th ACM Computer Science in Cars Symposium. CSCS 2021, Association for Computing Machinery, New York, NY, USA (2021). <https://doi.org/10.1145/3488904.3493385>
41. Kern, D., Lauser, T., Krauß, C.: Integrating privacy into the electric vehicle charging architecture. *Proceed. Priv. Enhan. Technol.* **3**, 140–158 (2022)
42. Lowe, G.: [A hierarchy of authentication specifications](#). In: Computer Security Foundations Workshop, pp. 31–43. IEEE (1997)
43. Macaulay, T., Henderson, R.: [Cryptographic Agility in Practice: Emerging Use-Cases](#). Infosec Global (2019)
44. Malina, L., Ricci, S., Dzurenda, P., Smekal, D., Hajny, J., Gerlich, T.: Towards practical deployment of post-quantum cryptography on constrained platforms and hardware-accelerated platforms. In: Simion, E., Géraud-Stewart, R. (eds.) SecITC 2019. LNCS, vol. 12001, pp. 109–124. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-41025-4_8
45. Mathilde, R., Aymeric, G., Yolán, R.: PQ-WireGuard: we did it again. <https://csrc.nist.gov/CSRC/media/Presentations/pq-wireguard-we-did-it-again/images-media/session-5-raynal-pq-wireguard.pdf> (2021)

46. McGrew, D., Curcio, M., Fluhrer, S.: [Leighton-Micali Hash-Based Signatures](#). RFC 8554 (2019)
47. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 696–701. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_48
48. Mosca, M., Piani, M.: [2021 Quantum Threat Timeline Report](#). Tech. rep., Global Risk Institute (2022)
49. NIST: Post-Quantum Cryptography PQC — Selected Algorithms 2022. <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>
50. NIST: Project: Post-Quantum Cryptography (2017). <https://csrc.nist.gov/Projects/post-quantum-cryptography>
51. OCA: Open Charge Point Protocol 2.0.1 - Part 2 - Specification. Open standard, Open Charge Alliance, Arnhem, Netherlands (3 2020). <https://www.openchargealliance.org/protocols/ocpp-201/>
52. Oder, T., Speith, J., Höltgen, K., Güneysu, T.: Towards practical microcontroller implementation of the signature scheme Falcon. In: Ding, J., Steinwandt, R. (eds.) PQCrypto 2019. LNCS, vol. 11505, pp. 65–80. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25510-7_4
53. Ott, D., Peikert, C., other workshop participants: Identifying Research Challenges in Post Quantum Cryptography Migration and Cryptographic Agility. arXiv preprint [arXiv:1909.07353](https://arxiv.org/abs/1909.07353) (2019)
54. Paquin, C., Stebila, D., Tamvada, G.: [Benchmarking Post-Quantum Cryptography in TLS](#). Cryptology ePrint Archive, Report 2019/1447 (2019)
55. Paul, S., Scheible, P.: Towards post-quantum security for cyber-physical systems: integrating PQC into industrial M2M communication. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12309, pp. 295–316. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59013-0_15
56. Ravi, P., Sundar, V.K., Chattopadhyay, A., Bhasin, S., Easwaran, A.: [Authentication Protocol for Secure Automotive Systems: Benchmarking Post-Quantum Cryptography](#). In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2020)
57. Renesas Electronics Corporation: R-Car Automotive System-on-Chips (SoCs) (2022). https://www.renesas.com/us/en/products/automotive-products/automotive-system-chips-socs#parametric_options
58. Shor, P.W.: [Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer](#). SIAM J. Comput. **26**(5), 1484–1509 (1997)
59. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: [Post-Quantum Authentication in TLS 1.3: A Performance Study](#). In: Network and Distributed System Security Symposium. Internet Society (2020)
60. Smith, M., Castellano, J.: [Costs Associated With Non-Residential Electric Vehicle Supply Equipment: Factors to consider in the implementation of electric vehicle charging stations](#). U.S. Department of Energy Vehicle Technologies Office (2015)
61. Smyslov, V.: Intermediate exchange in the IKEv2 protocol. IETF draft (2021)
62. Stebila, D., Fluhrer, S., Gueron, S.: [Hybrid key exchange in TLS 1.3](#). IETF draft (2020)
63. Stebila, D., Mosca, M.: Post-quantum key exchange for the internet and the open quantum safe project. In: Avanzi, R., Heys, H. (eds.) SAC 2016. LNCS, vol. 10532, pp. 14–37. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_2
64. Texas Instruments: DRA745 – Infotainment Applications Processor (2019). <https://www.ti.com/product/DRA745>

65. V2G Clarity: Reference Implementation Supporting the Evolution of the Vehicle-2-Grid communication interface (RISE V2G) (2020). <https://github.com/V2GClarity/RISE-V2G>
66. Vector: vSECC – Communication Controller for High Power Charging Stations V2.3 (2022). https://cdn.vector.com/cms/content/products/vSECC/Docs/vSECC_FactSheet_EN.pdf
67. Wang, W., Stöttinger, M.: [Post-Quantum Secure Architectures for Automotive Hardware Secure Modules](#). Cryptology ePrint Archive, Report 2020/026 (2020)
68. Wesemeyer, S., Newton, C.J., Treharne, H., Chen, L., Sasse, R., Whitefield, J.: Formal analysis and implementation of a TPM 2.0-based direct anonymous attestation scheme. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, pp. 784–798. ASIA CCS 2020, Association for Computing Machinery, New York, NY, USA (2020). <https://doi.org/10.1145/3320269.3372197>
69. Zhdanova, M., Urbansky, J., Hagemeyer, A., Zelle, D., Herrmann, I., Höffner, D.: Local power grids at risk – an experimental and simulation-based analysis of attacks on vehicle-to-grid communication. In: Proceedings of the 38th Annual Computer Security Applications Conference, pp. 42–55. ACSAC 2022, Association for Computing Machinery, New York, NY, USA (2022). <https://doi.org/10.1145/3564625.3568136>

Privacy-Preserving Protocols



Constant-Round Multiparty Private Function Evaluation with (Quasi-)Linear Complexities

Yongfeng Xu^{1,2}, Hanyu Jia¹, Xiangxue Li¹(✉), Qiang Li³, Yue Bao⁴,
and Xintian Hou⁴

¹ School of Software Engineering, East China Normal University,
Shanghai 200062, China
xxli@cs.ecnu.edu.cn

² Shanghai Key Laboratory of Privacy-Preserving Computation,
MatrixElements Technologies, Shanghai 201204, China

³ Institute of Cyber Science and Technology, Shanghai Jiaotong University,
Shanghai 200240, China
qiangli@sjtu.edu.cn

⁴ CATARC Software Testing (Tianjin) Co. Ltd., Tianjin 300300, China
{baoyue,houxintian}@catarc.ac.cn

Abstract. *Private function evaluation* (PFE) is a special case of secure multiparty computation. In multiparty PFE, the party P_1 holds its private n -variable function f and private input x_1 , while other parties P_i ($n \geq i \geq 2$) hold their private input x_i . All n participants can jointly evaluate the function f , and learn nothing from the interactions except the result $f(x_1, \dots, x_n)$ (known to a subset or all of the parties). The existing multiparty PFE protocols (e.g., Mohassel et al. at Eurocrypt'13 and Asiacypt'14) are with round complexity $O(g)$ (g is the circuit size) which makes them extremely unpractical. In this work, we propose for the first time constant-round multiparty PFE protocols that are secure against any number of corrupted parties under the semi-honest security model. We design our first construction from *oblivious evaluation of switching network* (OSN) protocol (Mohassel et al. at Eurocrypt'13), which only needs 9 rounds of interaction and can achieve quasi-linear communication and computation complexities (i.e., $O(ng \log(g))$). Our second construction is based on singly homomorphic encryption, which only needs 8 rounds of interaction and can achieve linear complexities. The OSN-based construction also benefits from the design trick that it only relies on symmetric operations (which makes it really efficient in actual executions). We further optimize our constructions by half-gate technology.

Keywords: Private function evaluation · Secure multiparty computation · Constant rounds · Linear complexity · Quasi-linear complexity

1 Introduction

SFE AND PFE. Standard Secure Multiparty Computation, also known as *Secure Function Evaluation* (SFE) [1, 18], refers to the joint evaluation of a public function f by n parties P_1, \dots, P_n using their private inputs x_i , and at the end the parties will learn nothing except the function output $f(x_1, \dots, x_n)$. *Private Function Evaluation* (PFE) is a special case of SFE [16]. In PFE, we assume that P_1 has private input data x_1 and private function f (and the corresponding circuit C_f). Other parties P_i have private input data x_i ($i = 2, \dots, n$). Some public information on the circuit is generally assumed, including its size, the lengths of its inputs and outputs, etc. All parties can perform the joint evaluation of the function (i.e., the circuit) and some of them (or all of them) would get the final outputs. No privacy leakage shall occur in the interactions among the parties. Compared with SFE, PFE provides more appealing privacy features, which might make it more meaningful (than SFE) in real-world scenarios. For example, algorithm service providers could use PFE to protect specific implementations of their algorithms from being revealed to their users. In recent years, some special- or general-purpose PFE protocols have been designed or implemented [3, 6, 16, 26]. The general-purpose PFE protocols are mainly divided into two types.

UC-BASED PFE. A universal circuit [27] U_g takes as input a boolean circuit C_f (containing at most g gates) and the party P_i takes private input $x_i, i = 1, \dots, n$. The final output is $f(x_1, \dots, x_n)$. As universal circuit size is the main factor that affects protocol efficiency, many attempts are made on optimizing the size of the universal circuit [12, 18, 20]. Liu et al. [21] achieved at Crypto 2021 the asymptotic circuit size $12g \log(g)$, which is the current state-of-the-art.

NON-UC PFE: 2-PARTY. The second type is to avoid the use of universal circuit. In [16], Katz et al. constructed a constant-round 2-party PFE (2-PFE) protocol with linear complexity under semi-honest security model based on garbled circuit [33] and homomorphic encryption. In [13], Holz et al. partially optimized the 2-PFE protocol [16] and implemented their protocol by using the ABY framework [9]. In [23], Mohassel et al. proposed a PFE framework under semi-honest security model, and constructed the *oblivious evaluation of switching networks* (OSN) protocol to realize *oblivious extended permutation* (OEP). They got a constant-round 2-PFE protocol based on the framework. The framework of [23] has later been extended to the malicious security model in [24]. In [7], Bingöl et al. optimized the 2-PFE protocol [23] by using half-gate technology [34] and removing the extra overhead of the special OSN construction in the 2-PFE protocol, which reduces the communication overhead by about 40%. Recently, Biçer et al. [6] proposed a constant-round 2-PFE protocol under the semi-honest security model based on asymmetric cryptographic primitives (i.e., Decisional Diffie-Hellman assumption, DDH). Their approach introduces reusability feature, saying that communication complexity could be significantly reduced from the second execution of the protocol. In [22], Liu et al. proposed the first constant-round 2-PFE protocol under the malicious security model, which can be seen as a malicious version of [6]. The protocol is also based on the DDH assumption and thereby supports reusability feature.

NON-UC PFE: MULTIPARTY. Compared with 2-party PFE (excluding the usage of universal circuit), there are relatively few researches in multiparty PFE (n -PFE). Most 2-party PFE protocols are based on standard garbled circuit [19] which works only in the two-party setting. Although some work [23, 24] proposed PFE frameworks, they are constructed in a similar way to the GMW paradigm [11], and if multiparty PFE protocols are constructed by using these frameworks, multiple rounds of communication are required (linear in the size of the circuit). This type of protocol usually has a good performance in the LAN settings, but the performance in the WAN settings is usually relatively poor due to the communication rounds. On the other hand, the communication on the wide area network is closer to the real-world scenarios, and in [4] Beaver et al. pointed out that the number of rounds is the most valuable resource. So it is valuable to implement constant-round multiparty PFE protocols.

We observe that it seems possible to use distributed garbling technique in PFE protocols. Some SFE protocols are constructed by using different distributed garbling fashion [4, 5, 8, 30, 31]. Note that the distributed garbling fashion in these SFE protocols can not be directly used in PFE protocols. For example, Ben-Efraim et al. constructed in [5] a constant-round multiparty SFE protocol based on the BMR protocol [4], where each party performs as both garbler and evaluator. However, for PFE protocols based on garbled circuit, there exists only one evaluator, i.e., the owner of the function f . In [8], Choi et al. proposed a distributed garbling fashion under the semi-honest security model and therein there is only one evaluator P_1 . The general idea is that all garblers P_i ($i = 2, \dots, n$) are responsible for jointly constructing the garbled circuit, and P_1 only needs to evaluate the garbled circuit. This is secure in the SFE protocol under the semi-honest security model. However, when we implement a similar garbling fashion in a multiparty PFE protocol, we note that partial information of the function f could be leaked once $n - 1$ garblers were corrupted. We observe that this problem can be solved as long as P_1 also participates in constructing the garbled circuit. In [30, 31], Wang et al. proposed a similar distributed garbling fashion under the malicious security model through the complex functionalities \mathcal{F}_{aAND} and \mathcal{F}_{abit} . In their protocols, the purpose of P_1 participating in constructing the garbled circuit is to implement a malicious SFE protocol without using the “cut-and-choose” technique, while in our protocol it is to ensure that P_1 can correctly evaluate the garbled circuit and the privacy of the function f .

OUR CONTRIBUTION. In this work, we mainly focus on the general-purpose n -PFE protocols that are not based on universal circuit. We propose for the first time constant-round multiparty PFE protocols under the semi-honest security model that are secure for any number of corrupted parties based on the garbled circuit in the distributed garbling fashion. We use OT protocols to achieve the distributed garbling fashion which is a variant of the garbling scheme proposed by Choi et al. in [8]. Our construction based on singly homomorphic encryption only costs 8 rounds of communication and can achieve linear communication and computation complexities. So the total communication overhead of this construction is relatively small, but it will cost $O(n\ell)$ asymmetric operations. Our construction based on OSN protocol will cost 9 rounds of communication, and the total communication overhead is relatively large, but it basically only relies

on symmetric operations (faster in actual executions). Compared with other 2-PFE protocols [23,34] that rely on symmetric operations, the communication overhead of our OSN-based protocol is about $1.3\times$ that of [23] and $2.3\times$ that of [34] at $n = 3$. In Sect. 4.2, we use a special half-gate technique to further optimize our multiparty PFE protocols, the communication overhead at each non-output gate and each output gate is reduced by k bits and $2k$ bits respectively. Section 5 will present detailed analysis of the concrete efficiency.

2 Notations and Preliminaries

We use k to denote computational security parameters, $=$ to denote equality, $:=$ to denote assignment operator. $lsb(s)$ represents the last bit of string s . We use n to denote the number of participants, f the function, C_f the bool circuit corresponding to f , g the number of gates in C_f , I the number of input wires of C_f , O the number of output wires of C_f . $[n]$ represents the set $\{1\dots n\}$, $|S|$ the size of set S , and \mathcal{C} the set of all corrupted parties. \mathcal{H} denotes the set of all honest parties. H is a hash function modeled as a random oracle, l is a positive integer. Usually in PFE protocols based on the garbled circuit, we assume that the type of all bool gates in the circuit C_f is *NAND* to avoid the necessity of hiding the gate functionality [6, 7, 16, 22, 23].

2.1 Outgoing Wires and Ingoing Wires

In the PFE protocols, all wires of the circuit are mainly divided into two sets OW (outgoing wires) and IW (ingoing wires). The OW set contains all the input wires of the circuit C_f and the output wires of non-output gates, and the IW set contains the input wires of each gate in the circuit. We have $M := |OW| = I + g - O$, $N := |IW| = 2g$. The mapping relationship $\pi : \{1, \dots, M\} \rightarrow \{1, \dots, N\}$ between OW and IW can represent the topology of the entire circuit C_f . For $i \in OW$ and $j \in IW$, $\pi(i) = j$ says that for “outgoing wire” OW_i , there is “ingoing wire” IW_j connected to OW_i . Note that since the fan-out of each gate can be more than one, π is not a function. We have however that π^{-1} is a function and $\pi^{-1}(j) = i$ says that for “ingoing wire” IW_j , there is only one “outgoing wire” OW_i connected to it. Figure 1 demonstrates an example circuit and its corresponding mapping.

2.2 Public Info

In PFE protocols, although the circuit C_f is the private data of P_1 , there do exist some public information, including the following [6, 16, 23]:

- (1) I , the number of input bits of the circuit C_f ;
- (2) O , the number of output bits of the circuit C_f ;
- (3) g , the number of gates in the circuit C_f ;
- (4) the “ingoing wire” and “outgoing wire” indices that belong to each gate;
- (5) the “outgoing wire” indices corresponding to each party’s input bits.

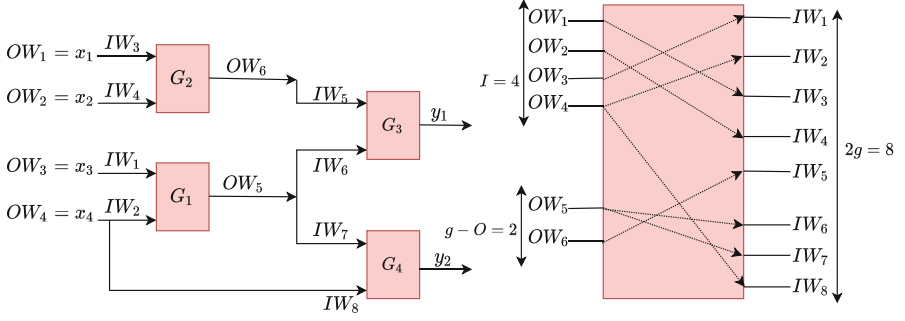


Fig. 1. A circuit C_f of function f (left) and a mapping π of C_f (right)

For a gate i , we use IW_{2i-1} and IW_{2i} to represent the “ingoing wire” corresponding to its input wires, and OW_{I+i} to represent the “outgoing wire” corresponding to its output wire. $\{OW_1, \dots, OW_I\}$ is the “outgoing wire” set corresponding to the input wires of the circuit. For convenience, we use $(\alpha, \beta, \gamma, NAND)$ to represent a $NAND$ gate in the SFE protocol. Herein, α and β represent the indices of the input wires of the gate, and γ represents the index of the output wire. $(IW_\alpha, IW_\beta, OW_\gamma, NAND)$ represents a $NAND$ gate in the PFE protocol, where IW_α and IW_β represent the indices of the “ingoing wire” corresponding to the input wires of the gate. OW_γ represents the index of the “outgoing wire” corresponding to the output wire of the gate, and also represents the index of the “outgoing wire” which is connected to the “ingoing wire” IW_γ , i.e., $\pi^{-1}(IW_\gamma) = OW_\gamma$.

2.3 EP and OEP

A mapping $\pi' : \{1, \dots, N\} \rightarrow \{1, \dots, N\}$ is a permutation if it is a bijection, while for a mapping $\pi : \{1, \dots, M\} \rightarrow \{1, \dots, N\}$ ($M \leq N$) is called an *Extended permutation* (EP), and the specific definition of extended permutation is as follows.

Definition 1 (Extended Permutation). For positive integers M and N , we call a mapping $\pi : \{1, \dots, M\} \rightarrow \{1, \dots, N\}$ ($M \leq N$) an *extended permutation* (EP) if for every $j \in \{1, \dots, N\}$, there is exactly one $i \in \{1, \dots, M\}$ such that $\pi(i) = j$. We often denote i by $\pi^{-1}(j)$.

The oblivious computation of an EP is *oblivious extended computation* (OEP). The specific definition of a 2-OEP is as follows.

Definition 2 (2-OEP). P_1 holds an extended permutation (EP) $\pi : \{1, \dots, M\} \rightarrow \{1, \dots, N\}$ and a blinding vector $\vec{t} = (t_1, \dots, t_N)$. P_2 holds the input vector $\vec{x} = (x_1, \dots, x_M)$. Both parties perform the OEP protocol, and at the end of the protocol P_2 learns $(x_{\pi^{-1}(1)} \oplus t_1, \dots, x_{\pi^{-1}(N)} \oplus t_N)$, P_1 learns nothing. t_i and x_i are both l -bit string.

2.4 OSN

Here we briefly describe OSN protocol, and the Appendix A gives a detailed description. In [23], Mohassel and Sadeghian implemented a constant-round 2-OEP protocol called as *oblivious evaluation of a switching network* (OSN) protocol. The OSN protocol is mainly composed of two components: (1) use a switching network to implement *extended permutation*, and (2) apply OT protocols to an oblivious evaluation of the switching network.

A switching network is a set of interconnected switches that takes N inputs and a set of selection bits, and outputs N values. Each switch in the network accepts two l -bit strings as inputs and outputs two l -bit strings. In order to implement the EP, the entire switching network is composed of $(2N \log(N) - N + 1)$ switches. In order to obliviously evaluate the entire switching network, it needs to cost a 1-out-of-2 OT at each switch. So the OSN protocol needs to cost $(2N \log(N) - N + 1)$ OT and all OT protocols can be executed in parallel. The protocol costs 3 rounds of communication. Figure 2(a) describes the corresponding functionality \mathcal{F}_{2-OSN} . Executing the 2-OSN protocol $n - 1$ times would lead to n -OSN protocol (see Fig. 2(b)).

Functionality \mathcal{F}_{2-OSN}	Functionality \mathcal{F}_{n-OSN}
<p>Inputs: The functionality receives (input, P_1, π, \vec{t}) from P_1 where $\pi : \{1, \dots, M\} \rightarrow \{1, \dots, N\}$, $\vec{t} = (t_1, \dots, t_N)$, $t_i \in \{0, 1\}^l$, and (input, P_2, \vec{x}) from P_2 where $\vec{x} = (x_1, \dots, x_M)$, $x_i \in \{0, 1\}^l$.</p> <p>Outputs: The functionality outputs $(x_{\pi^{-1}(1)} \oplus t_1, \dots, x_{\pi^{-1}(N)} \oplus t_N)$ to P_2, and nothing to P_1.</p>	<p>Inputs: For $i \in [n]$, $i \neq 1$ the functionality receives (input, $P_1, \pi, \{\vec{t}_i\}_{i \geq 2}$) from P_1 where $\pi : \{1, \dots, M\} \rightarrow \{1, \dots, N\}$, $\vec{t}_i = (t_1^i, \dots, t_N^i)$, $t_j^i \in \{0, 1\}^l$, and (input, P_i, \vec{x}_i) from P_i where $\vec{x}_i = (x_1^i, \dots, x_M^i)$, $x_j^i \in \{0, 1\}^l$.</p> <p>Outputs: The functionality outputs $(x_{\pi^{-1}(1)}^i \oplus t_1^i, \dots, x_{\pi^{-1}(N)}^i \oplus t_N^i)$ to P_i, and nothing to P_1.</p>
(a)	(b)

Fig. 2. Functionality \mathcal{F}_{2-OSN} and Functionality \mathcal{F}_{n-OSN}

3 Overview

In this section, we will give a high-level overview of our constant-round multi-party PFE protocols. Section 4 will say more details of our protocols.

In order to understand our protocols more clearly, we first review the construction of garbled tables in the standard 2-party garbled circuit, similar to the description in [29] by Wang et al. In a standard garbled circuit, each wire α of the circuit is associated with a random “mask bit” λ_α known only to the garbler P_2 . P_2 generates a key label $L_{\alpha,0}$ for each wire α that corresponds to masked value “0” and $L_{\alpha,1} = L_{\alpha,0} \oplus \Delta$ that corresponds to masked value “1”. The masked value represents the result of the XOR operation between the truth value and the mask bit on the wire, e.g., if the truth value on the wire α is x , then the masked value $A_\alpha = x \oplus \lambda_\alpha$. For a *NAND* gate $(\alpha, \beta, \gamma, \text{NAND})$,

the corresponding truth values and mask bits on the wires α, β, γ are x, y, z and $\lambda_\alpha, \lambda_\beta, \lambda_\gamma$ respectively. When evaluating the gate, the evaluator P_1 has the masked value $A_\alpha = x \oplus \lambda_\alpha$ and the corresponding key label L_{α, A_α} on wire α , and the masked value $A_\beta = y \oplus \lambda_\beta$ and the corresponding key label L_{β, A_β} on wire β . P_1 can not know the truth values x and y because P_1 does not know λ_α and λ_β . P_1 evaluates the row $(2A_\alpha + A_\beta)$ of the garbled table and gets the result $(A_{\gamma, A_\alpha A_\beta}, L_{\gamma, A_{\gamma, A_\alpha A_\beta}})$. $A_{\gamma, A_\alpha A_\beta} = z \oplus \lambda_\gamma$ where $z = NAND(x, y)$, so P_1 correctly evaluates the $NAND$ gate and P_1 does not know the truth value z on the wire γ . Figure 3 gives the specific form of the garbled table.

Λ_α	Λ_β	truth table	garbled table
0	0	$\Lambda_{\gamma,00} = (\lambda_\alpha \wedge \lambda_\beta) \oplus 1 \oplus \lambda_\gamma$	$H(L_{\alpha,0}, L_{\beta,0}, \gamma, 00) \oplus (\Lambda_{\gamma,00}, L_{\gamma, \Lambda_{\gamma,00}})$
0	1	$\Lambda_{\gamma,01} = (\lambda_\alpha \wedge \bar{\lambda}_\beta) \oplus 1 \oplus \lambda_\gamma$	$H(L_{\alpha,0}, L_{\beta,1}, \gamma, 01) \oplus (\Lambda_{\gamma,01}, L_{\gamma, \Lambda_{\gamma,01}})$
1	0	$\Lambda_{\gamma,10} = (\bar{\lambda}_\alpha \wedge \lambda_\beta) \oplus 1 \oplus \lambda_\gamma$	$H(L_{\alpha,1}, L_{\beta,0}, \gamma, 10) \oplus (\Lambda_{\gamma,10}, L_{\gamma, \Lambda_{\gamma,10}})$
1	1	$\Lambda_{\gamma,11} = (\bar{\lambda}_\alpha \wedge \bar{\lambda}_\beta) \oplus 1 \oplus \lambda_\gamma$	$H(L_{\alpha,1}, L_{\beta,1}, \gamma, 11) \oplus (\Lambda_{\gamma,11}, L_{\gamma, \Lambda_{\gamma,11}})$

Fig. 3. The garbled table of $NAND$ gate in the standard garbled circuit.

Next we show how to construct the garbled table in distributed garbling fashion. We only need to use OT protocols. Compared with [8], our implementation is clearer and easier to understand, and we also analyze the concrete efficiency. Another difference is that P_1 needs to participate in constructing the garbled circuit, but P_1 is not garbler which is also different from BMR circuit [5]. There are n participants, P_1 is the evaluator and P_i ($i = 2, \dots, n$) are the garblers. All n parties must participate in constructing the garbled table of each gate. The “mask bit” λ_α on each wire α in the circuit is no longer held by the garbler P_2 alone, but each of n parties holds a secret share of λ_α , namely λ_α^i for P_i , where $\lambda_\alpha = \bigoplus_{i=1}^n \lambda_\alpha^i$. It can be realized by randomly selecting a bit by each participant. Each garbler P_i ($i \geq 2$) generates for each wire α a pair of key labels $(L_{\alpha,0}^i, L_{\alpha,1}^i)$ corresponding to the masked value “0” and “1” and a random value $\Delta_i \in \{0, 1\}^k$, where $L_{\alpha,1}^i = L_{\alpha,0}^i \oplus \Delta_i$. For $\Lambda_{\gamma,00} = (\lambda_\alpha \wedge \lambda_\beta) \oplus 1 \oplus \lambda_\gamma = (\lambda_\alpha^1 \oplus \dots \oplus \lambda_\alpha^n)(\lambda_\beta^1 \oplus \dots \oplus \lambda_\beta^n) \oplus 1 \oplus \lambda_\gamma$ encrypted by the garbled table, each garbler P_i only holds the secret shares of $\lambda_\alpha, \lambda_\beta, \lambda_\gamma$. Thus, P_i can not directly construct $\Lambda_{\gamma,00}$, but P_i can construct the secret share of $\Lambda_{\gamma,00}$. For $\lambda_\alpha \wedge \lambda_\beta = (\lambda_\alpha^1 \oplus \dots \oplus \lambda_\alpha^n)(\lambda_\beta^1 \oplus \dots \oplus \lambda_\beta^n)$, P_i holds $\lambda_\alpha^i \lambda_\beta^i$.

For $\lambda_\alpha^i(\lambda_\beta^1 \oplus \dots \oplus \lambda_\beta^n)$, P_i can perform 1-out-of-2 OT protocol with each party P_j ($j \in [n], j \neq i$). The input of P_i in the OT protocol is λ_α^i and that of P_j is $(s, s \oplus \lambda_\beta^j)$, where s is a random bit selected by P_j . Finally, P_i learns $s \oplus \lambda_\alpha^i \lambda_\beta^j$, P_j learns s , and both parties get the secret shares of $\lambda_\alpha^i \lambda_\beta^j$. The OT implementing such function is called bitOT, and we describe the corresponding functionality \mathcal{F}_{bitOT} in Fig. 4(a). So each of the n parties can learn a secret share of $\lambda_\alpha^i(\lambda_\beta^1 \oplus \dots \oplus \lambda_\beta^n)$. For $i, j \in [n]$, if each pair of parties (P_i, P_j) ($i \neq j$) runs the functionality \mathcal{F}_{bitOT} , each party can learn a secret share of $\lambda_\alpha \wedge \lambda_\beta$ and then compute a secret share of $\Lambda_{\gamma,00}$. $\Lambda_{\gamma,00}, \Lambda_{\gamma,01}, \Lambda_{\gamma,10}$, and $\Lambda_{\gamma,11}$ have the following relationship: $\Lambda_{\gamma,01} = \Lambda_{\gamma,00} \oplus \lambda_\alpha$, $\Lambda_{\gamma,10} = \Lambda_{\gamma,00} \oplus \lambda_\beta$, $\Lambda_{\gamma,11} = \Lambda_{\gamma,00} \oplus \lambda_\alpha \oplus \lambda_\beta \oplus 1$. For $j \in [n]$, P_j can use the secret shares of $\lambda_\alpha \wedge \lambda_\beta, \lambda_\alpha, \lambda_\beta, \lambda_\gamma$ to construct the secret shares of $\Lambda_{\gamma,00}, \Lambda_{\gamma,01}, \Lambda_{\gamma,10}, \Lambda_{\gamma,11}$ locally. For the key label $L_{\gamma, \Lambda_{\gamma,00}}^i = L_{\gamma,0}^i \oplus \Lambda_{\gamma,00} \Delta_i$ encrypted by each garbler P_i ($i \geq 2$) in its garbled table, where $\Lambda_{\gamma,00} \Delta_i = (\Lambda_{\gamma,00}^1 \oplus \dots \oplus \Lambda_{\gamma,00}^n) \Delta_i$, each party can also construct a secret share of $L_{\gamma, \Lambda_{\gamma,00}}^i$. After performing the functionality \mathcal{F}_{bitOT} , P_i holds $\Lambda_{\gamma,00}^i$ and computes $\Lambda_{\gamma,00}^i \Delta_i$.

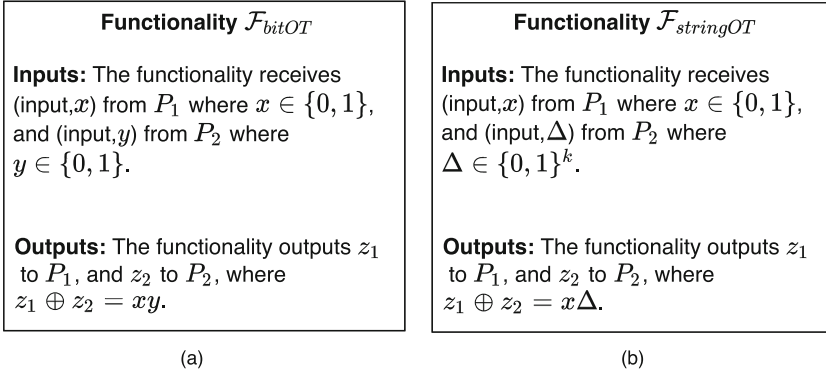


Fig. 4. bitOT and stringOT

For $j \neq i$, P_i can also perform a 1-out-of-2 OT protocol with P_j to get the secret share of $\Lambda_{\gamma,00}^j \Delta_i$. The input of P_j in the OT protocol is $\Lambda_{\gamma,00}^j$ and that of P_i is $(S, S \oplus \Delta_i)$, where S is a random bit string selected by P_i . Finally, P_j learns $S \oplus \Lambda_{\gamma,00}^j \Delta_i$, P_i learns S , and both parties get the secret share of $\Lambda_{\gamma,00}^j \Delta_i$. The OT that implements such a function is called stringOT, and we describe the corresponding functionality $\mathcal{F}_{stringOT}$ in Fig. 4(b). After performing the functionality $\mathcal{F}_{stringOT}$, n parties can learn the secret shares of $\Lambda_{\gamma,00} \Delta_i$, so they can compute the secret shares of $L_{\gamma, \Lambda_{\gamma,00}}^i$ locally:

- P_i holds $(L_{\gamma, \Lambda_{\gamma,00}}^i)^i = L_{\gamma,0}^i \oplus (\Lambda_{\gamma,00} \Delta_i)^i$;
- P_j ($j \neq i$) holds $(L_{\gamma, \Lambda_{\gamma,00}}^i)^j = (\Lambda_{\gamma,00} \Delta_i)^j$.

The operations of constructing the secret shares of $L_{\gamma, \Lambda_{\gamma, 00}}^i, L_{\gamma, \Lambda_{\gamma, 01}}^i, L_{\gamma, \Lambda_{\gamma, 10}}^i, L_{\gamma, \Lambda_{\gamma, 11}}^i$ are same as $L_{\gamma, \Lambda_{\gamma, 00}}^i$. Thus, n parties can learn the relevant secret shares through the functionalities \mathcal{F}_{bitOT} and $\mathcal{F}_{stringOT}$, and Fig. 5 shows the garbled table of each gate constructed by any garbler P_i ($i \geq 2$) and secret shares held by P_1 .

Λ_α	Λ_β	P_1 's share of garbled table	P_i 's share of garbled table
0	0	$(\Lambda_{\gamma, 00}^1, \{(L_{\gamma, \Lambda_{\gamma, 00}}^j)^1\}_{j \neq 1})$	$H(L_{\alpha, 0}^i, L_{\beta, 0}^i, \gamma, 00) \oplus (\Lambda_{\gamma, 00}^i, (L_{\gamma, \Lambda_{\gamma, 00}}^j)^i, \{(L_{\gamma, \Lambda_{\gamma, 00}}^j)^i\}_{j \neq i, 1})$
0	1	$(\Lambda_{\gamma, 01}^1, \{(L_{\gamma, \Lambda_{\gamma, 01}}^j)^1\}_{j \neq 1})$	$H(L_{\alpha, 0}^i, L_{\beta, 1}^i, \gamma, 01) \oplus (\Lambda_{\gamma, 01}^i, (L_{\gamma, \Lambda_{\gamma, 01}}^j)^i, \{(L_{\gamma, \Lambda_{\gamma, 01}}^j)^i\}_{j \neq i, 1})$
1	0	$(\Lambda_{\gamma, 10}^1, \{(L_{\gamma, \Lambda_{\gamma, 10}}^j)^1\}_{j \neq 1})$	$H(L_{\alpha, 1}^i, L_{\beta, 0}^i, \gamma, 10) \oplus (\Lambda_{\gamma, 10}^i, (L_{\gamma, \Lambda_{\gamma, 10}}^j)^i, \{(L_{\gamma, \Lambda_{\gamma, 10}}^j)^i\}_{j \neq i, 1})$
1	1	$(\Lambda_{\gamma, 11}^1, \{(L_{\gamma, \Lambda_{\gamma, 11}}^j)^1\}_{j \neq 1})$	$H(L_{\alpha, 1}^i, L_{\beta, 1}^i, \gamma, 11) \oplus (\Lambda_{\gamma, 11}^i, (L_{\gamma, \Lambda_{\gamma, 11}}^j)^i, \{(L_{\gamma, \Lambda_{\gamma, 11}}^j)^i\}_{j \neq i, 1})$

Fig. 5. The secret shares of garbled table in n parties

Through above analysis, we can know the OT overhead of each party (as sender and receiver) in constructing garbled table of each bool gate.

Cost. P_1 needs to perform $2(n - 1)$ bitOT (to construct secret shares of $\lambda_\alpha \lambda_\beta$), and $4(n - 1)$ stringOT (to construct secret shares of $\{L_{\gamma, \Lambda_{\gamma, uv}}^j\}_{j \neq 1}$, $u, v \in \{0, 1\}$). Each garbler P_i ($i \geq 2$) needs to perform $2(n - 1)$ bitOT (to construct secret shares of $\lambda_\alpha \lambda_\beta$), and $8n - 12$ stringOT (to construct secret shares of $\{L_{\gamma, \Lambda_{\gamma, uv}}^j\}_{j \neq 1}$, $u, v \in \{0, 1\}$). We also have the following observations.

1. $L_{\gamma, \Lambda_{\gamma, 00}}^i = L_{\gamma, 0}^i \oplus \Lambda_{\gamma, 00} \Delta_i$, all n parties can learn the secret shares of $\Lambda_{\gamma, 00} \Delta_i$ by performing stringOT and thereby the secret shares of $L_{\gamma, \Lambda_{\gamma, 00}}^i$.
2. $L_{\gamma, \Lambda_{\gamma, 01}}^i = L_{\gamma, 0}^i \oplus \Lambda_{\gamma, 01} \Delta_i = L_{\gamma, 0}^i \oplus (\Lambda_{\gamma, 00} \oplus \lambda_\alpha) \Delta_i$, on the basis of holding secret shares of $\Lambda_{\gamma, 00} \Delta_i$, all n parties only need to learn secret shares of $\lambda_\alpha \Delta_i$ by performing stringOT, and can thereby learn the secret shares of $L_{\gamma, \Lambda_{\gamma, 01}}^i$.
3. $L_{\gamma, \Lambda_{\gamma, 10}}^i = L_{\gamma, 0}^i \oplus \Lambda_{\gamma, 10} \Delta_i = L_{\gamma, 0}^i \oplus (\Lambda_{\gamma, 00} \oplus \lambda_\beta) \Delta_i$, on the basis of holding the secret shares of $\Lambda_{\gamma, 00} \Delta_i$, all n parties only need to learn the secret shares of $\lambda_\beta \Delta_i$ by performing stringOT, thereby learning the secret shares of $L_{\gamma, \Lambda_{\gamma, 10}}^i$.
4. $L_{\gamma, \Lambda_{\gamma, 11}}^i = L_{\gamma, 0}^i \oplus \Lambda_{\gamma, 11} \Delta_i = L_{\gamma, 0}^i \oplus (\Lambda_{\gamma, 00} \oplus \lambda_\alpha \oplus \lambda_\beta \oplus 1) \Delta_i$, on the basis of holding the secret shares of $\Lambda_{\gamma, 00} \Delta_i$, $\lambda_\alpha \Delta_i$, $\lambda_\beta \Delta_i$, each party can construct a secret share of $L_{\gamma, \Lambda_{\gamma, 11}}^i$ locally.

Therefore, there is no need to cost extra stringOT for constructing the secret shares of $L_{\gamma, \Lambda_{\gamma, 11}}^i$. So P_1 only needs to cost $3(n - 1)$ stringOT, and P_i only needs to cost $6n - 9$ stringOT.

Through the above analysis, we know how the parties cooperate to construct the garbled table of each bool gate in the n -party case. But this is not enough to construct our multiparty PFE protocols, as the circuit C_f in the PFE protocols is the private data of P_1 , and each garbler P_i does not know the topology of

C_f . Namely, each bool gate is isolated and irrelevant for P_i . P_i can not correctly construct the garbled table for each gate in the circuit C_f .

In Sect. 2.1, we already define the “outgoing wire” and “ingoing wire”, and the mapping relationship $\pi : \{1, \dots, M\} \rightarrow \{1, \dots, N\}$ between OW and IW which represents the topology of the circuit C_f . P_1 can learn π according to the circuit C_f . In order for all garblers P_i ($i \geq 2$) to construct garbled tables correctly, P_1 needs to use π to help P_i connect all isolated gates in the topological order of C_f . But this process should be oblivious to P_i , ensuring that P_i can learn the necessary information to construct the garbled tables correctly and that the information of π will not be leaked. Now, the OEP protocol could be used to achieve such a goal.

More precisely, each garbler P_i generates a key label and a secret share of mask bit for each “outgoing wire”, i.e., $(L_{1,0}^i, \dots, L_{M,0}^i)$, $(\lambda_1^i, \dots, \lambda_M^i)$, and uses them as the inputs of the OEP protocol. The inputs of P_1 are π and the blind vectors $\vec{T} = (T_1^i, \dots, T_N^i)$, $\vec{t} = (t_1^i, \dots, t_N^i)$. After the OEP protocol, P_i learns the key labels $(L_{\pi^{-1}(1),0}^i \oplus T_1^i, \dots, L_{\pi^{-1}(N),0}^i \oplus T_N^i)$ and $(\lambda_{\pi^{-1}(1)}^i \oplus t_1^i, \dots, \lambda_{\pi^{-1}(N)}^i \oplus t_N^i)$ on all “ingoing wires”, and we simplify them to $(L_{1,0}^{i'}, \dots, L_{N,0}^{i'})$, $(\lambda_1^{i'}, \dots, \lambda_N^{i'})$. P_i computes $L_{j,1}^{i'} = L_{j,0}^{i'} \oplus \Delta_i$, $j \in [N]$. In this way, for each $NAND$ gate $gate_j$, P_i has the key labels and secret shares of mask bit on IW_{2j-1} , IW_{2j} , OW_{I+j} to construct its own garbled table (after performing \mathcal{F}_{bitOT} and $\mathcal{F}_{stringOT}$), and all garbled tables can be constructed correctly, because all “outgoing wire” and “ingoing wire” have been obliviously connected in topological order of C_f through OEP protocol. In the evaluation stage, for a gate $(IW_\alpha, IW_\beta, OW_\gamma, NAND)$ where $\pi^{-1}(IW_\alpha) = OW_\alpha$, $\pi^{-1}(IW_\beta) = OW_\beta$, $\pi^{-1}(IW_\gamma) = OW_\gamma$, P_1 initially has $(\Lambda_{OW_\alpha}, \{L_{OW_\alpha, \Lambda_{OW_\alpha}}^i\}_{i \neq 1})$ on OW_α and $(\Lambda_{OW_\beta}, \{L_{OW_\beta, \Lambda_{OW_\beta}}^i\}_{i \neq 1})$ on OW_β . P_1 computes $(\Lambda'_{IW_\alpha}, \{L_{IW_\alpha, \Lambda'_{IW_\alpha}}^{i'}\}_{i \neq 1})$ and $(\Lambda'_{IW_\beta}, \{L_{IW_\beta, \Lambda'_{IW_\beta}}^{i'}\}_{i \neq 1})$ by performing XOR operation between the key labels on OW_α , OW_β and corresponding blind values T according to the information of π . In fact, we have $\Lambda_{OW_\alpha} = \Lambda'_{IW_\alpha}$ and $\Lambda_{OW_\beta} = \Lambda'_{IW_\beta}$ (see the details in Sect. 4.1). P_1 evaluates the row $(2\Lambda'_{IW_\alpha} + \Lambda'_{IW_\beta})$ of $n-1$ garbled tables for this gate, and then reconstructs the masked value Λ_{OW_γ} and key labels $\{L_{OW_\gamma, \Lambda_{OW_\gamma}}^i\}_{i \neq 1}$ of the output wire OW_γ . P_1 repeats the above operations to evaluate the complete circuit according to the topology of C_f .

4 Main Protocol

Now we are ready to describe our protocols in detail. Section 4.1 will implement the constant-round multiparty PFE protocol based on \mathcal{F}_{n-OSN} . Section 4.2 will further optimize our PFE protocol by using half-gate technology. In Sect. 4.3, we will implement OEP protocol using singly homomorphic encryption.

4.1 Multiparty Constant-Round PFE

Our protocol is mainly divided into the preprocessing stage and the online stage. The detailed protocol description is shown in Fig. 6. For the step 4 in Fig. 6, in order to construct the garbled tables in distributed garbling, each party also needs to learn the secret share of the mask bit on each “incoming wire”. And in order to evaluate the garbled circuit correctly, for each “incoming wire” $IW_\alpha \in [N]$, we need to ensure $\lambda_{OW_\alpha} = \lambda'_{IW_\alpha}$ where $\pi^{-1}(IW_\alpha) = OW_\alpha$. If $\lambda_{OW_\alpha} \neq \lambda'_{IW_\alpha}$, when P_1 evaluates the gate $(IW_\alpha, IW_\beta, OW_\gamma, NAND)$, P_1 holds $(\Lambda_{OW_\alpha}, \{L^i_{OW_\alpha, \Lambda_{OW_\alpha}}\}_{i \neq 1})$, computes $\{L^i_{IW_\alpha, \Lambda_{OW_\alpha}} = L^i_{OW_\alpha, \Lambda_{OW_\alpha}} \oplus T^i_{IW_\alpha}\}_{i \neq 1}$ and Λ'_{IW_α} . If $\Lambda_{OW_\alpha} = \Lambda'_{IW_\alpha}$, it means that the truth value x on OW_α is not equal to that on IW_α because $\lambda_{OW_\alpha} \neq \lambda'_{IW_\alpha}$. If $\Lambda_{OW_\alpha} \neq \Lambda'_{IW_\alpha}$, Λ'_{IW_α} and $\{L^i_{IW_\alpha, \Lambda_{OW_\alpha}}\}_{i \neq 1}$ will conflict when P_1 evaluates the gabled table. So P_1 needs to compute $(\lambda_1^1, \dots, \lambda_N^1)$ as in step 4. Any $n-1$ parties can not learn λ_{OW_α} and λ'_{IW_α} , so the operations in step 4 also ensure the privacy of the circuit topology. Next, we will use Λ_α to represent the masked value on IW_α and OW_α . If we use the OT protocols in our protocol to directly implement the distributed garbling fashion proposed by Choi et al. [8], it will be more efficient. Because it is not necessary for P_1 to participate in constructing the garbled circuit, P_1 also does not need to generate the secret share of mask bit on each “outgoing wire”, i.e., $\lambda_{OW_\alpha} = \bigoplus_{i=2}^n \lambda^i_{OW_\alpha}$. However, in order to ensure $\lambda_{OW_\alpha} = \lambda'_{IW_\alpha}$, P_1 needs to ensure $(\bigoplus_{i \geq 2}^n t_j^i) = 0$ in the step 4. When all $n-1$ garblers are corrupted, they can learn λ'_{IW_α} and λ_{OW_α} , then they will learn part of the information of π .

In step 5, n parties can learn the secret shares of $\Lambda_{OW_\gamma, 00}$, $\Lambda_{OW_\gamma, 01}$, $\Lambda_{OW_\gamma, 10}$, $\Lambda_{OW_\gamma, 11}$ by performing a series of \mathcal{F}_{bitOT} . The secret shares held by each party are as follows.

Secret shares held by P_1

$$\Lambda_{\gamma, 00}^1 = (\lambda'_{IW_\alpha} \lambda'_{IW_\beta})^1 \oplus 1 \oplus \lambda^1_{OW_\gamma}$$

$$\Lambda_{\gamma, 01}^1 = \Lambda_{\gamma, 00}^1 \oplus \lambda^1_{IW_\alpha}$$

$$\Lambda_{\gamma, 10}^1 = \Lambda_{\gamma, 00}^1 \oplus \lambda^1_{IW_\beta}$$

$$\Lambda_{\gamma, 11}^1 = \Lambda_{\gamma, 00}^1 \oplus \lambda^1_{IW_\alpha} \oplus \lambda^1_{IW_\beta} \oplus 1$$

Secret shares held by P_i

$$\Lambda_{\gamma, 00}^i = (\lambda'_{IW_\alpha} \lambda'_{IW_\beta})^i \oplus \lambda^i_{OW_\gamma}$$

$$\Lambda_{\gamma, 01}^i = \Lambda_{\gamma, 00}^i \oplus \lambda^i_{IW_\alpha}$$

$$\Lambda_{\gamma, 10}^i = \Lambda_{\gamma, 00}^i \oplus \lambda^i_{IW_\beta}$$

$$\Lambda_{\gamma, 11}^i = \Lambda_{\gamma, 00}^i \oplus \lambda^i_{IW_\alpha} \oplus \lambda^i_{IW_\beta}$$

Now n parties can learn the secret shares of $L^i_{OW_\gamma, \Lambda_\gamma, 00}$, $L^i_{OW_\gamma, \Lambda_\gamma, 01}$, $L^i_{OW_\gamma, \Lambda_\gamma, 10}$, $L^i_{OW_\gamma, \Lambda_\gamma, 11}$ by performing a series of $\mathcal{F}_{stringOT}$ on the basis that each party holds the secret share of $\Lambda_{\gamma, 00}$. P_j ($j \neq i$) holds the following secret shares:

$$\begin{aligned} (L^i_{OW_\gamma, \Lambda_\gamma, 00})^j &= (\Lambda_{\gamma, 00} \Delta_i)^j \\ (L^i_{OW_\gamma, \Lambda_\gamma, 01})^j &= (\Lambda_{\gamma, 01} \Delta_i)^j = (\Lambda_{\gamma, 00} \Delta_i)^j \oplus (\lambda'_{IW_\alpha} \Delta_i)^j \\ (L^i_{OW_\gamma, \Lambda_\gamma, 10})^j &= (\Lambda_{\gamma, 10} \Delta_i)^j = (\Lambda_{\gamma, 00} \Delta_i)^j \oplus (\lambda'_{IW_\beta} \Delta_i)^j \\ (L^i_{OW_\gamma, \Lambda_\gamma, 11})^j &= (\Lambda_{\gamma, 11} \Delta_i)^j = (\Lambda_{\gamma, 00} \Delta_i)^j \oplus (\lambda'_{IW_\alpha} \Delta_i)^j \oplus (\lambda'_{IW_\beta} \Delta_i)^j \end{aligned} \quad (1)$$

Protocol \prod_{PFE}

Input: P_1 holds $x_{I_1} \in \{0,1\}^{I_1}$ and P_i ($i \geq 2$) holds $x_{I_i} \in \{0,1\}^{I_i}$. P_1 holds the function $f : \{0,1\}^{I_1} \times \dots \times \{0,1\}^{I_n} \rightarrow \{0,1\}^O$ and circuit C_f . All parties know $I, G, O, M = I + g - O, N = 2g$. The type of bool gates are all *NAND* in the circuit C_f .

Preprocessing:

1. P_1 learns the mapping relationship π between all "outgoing wire" and "ingoining wire" according to the circuit C_f .
2. $j \in [n]$, P_j selects a random value $\lambda_{OW_\alpha}^j$ as the secret share of λ_{OW_α} for each "outgoing wire" OW_α and a secret share λ_O^j for output wire O .
3. P_i ($i \geq 2$) generates a random value $\Delta_i \in \{0,1\}^k$, and generates a key label $L_{OW_\alpha,0}^i$ for each "outgoing wire" OW_α and a key label $L_{O,0}^i$ for output wire O . P_i computes $L_{OW_\alpha,1}^i = L_{OW_\alpha,0}^i \oplus \Delta_i$, $L_{O,1}^i = L_{O,0}^i \oplus \Delta_i$.
4. P_i ($i \geq 2$) performs functionality \mathcal{F}_{n-OSN} with P_1 , the inputs of P_i are $(\lambda_1^i, \dots, \lambda_M^i)$, $(L_{1,0}^i, \dots, L_{M,0}^i)$ which can be combined as $(L_{1,0}^i || \lambda_1^i, \dots, L_{M,0}^i || \lambda_M^i)$, and in order to make it easier to understand we will use the split form in our analyses. The inputs of P_i are $\pi_i, (t_1^i, \dots, t_N^i), (T_1^i, \dots, T_N^i)$ where $t^i \in \{0,1\}, T^i \in \{0,1\}^k$. The outputs of P_i are $(\lambda_{\pi^{-1}(1)}^i \oplus t_1^i, \dots, \lambda_{\pi^{-1}(N)}^i \oplus t_N^i)$, $(L_{\pi^{-1}(1),0}^i \oplus T_1^i, \dots, L_{\pi^{-1}(N),0}^i \oplus T_N^i)$. We simplify them to $(\lambda_1^i, \dots, \lambda_N^i)$, $(L_{1,0}^i, \dots, L_{N,0}^i)$. For $\pi^{-1}(IW_\alpha) = OW_\alpha$ ($IW_\alpha \in [N]$), $\lambda'_{IW_\alpha} = \lambda_{OW_\alpha}^i \oplus t'_{IW_\alpha}$, $L'_{IW_\alpha,0} = L_{OW_\alpha,0}^i \oplus T'_{IW_\alpha}$, P_i computes $L'_{IW_\alpha,1} = L'_{IW_\alpha,0} \oplus \Delta_i$. Since P_1 knows π , P_1 can perform the following computations to learn the secret share of mask bits on all "ingoining wires":

$$(\lambda_1^i, \dots, \lambda_N^i) = (\lambda_{\pi^{-1}(1)}^i \oplus (\bigoplus_{i \geq 2} t_1^i), \dots, \lambda_{\pi^{-1}(N)}^i \oplus (\bigoplus_{i \geq 2} t_N^i))$$

$$\lambda'_{IW_\alpha} \neq \lambda_{OW_\alpha}^i, \text{ but } \bigoplus_{i=1}^n \lambda_{OW_\alpha}^i = \lambda_{OW_\alpha} = \lambda'_{IW_\alpha} = \bigoplus_{i=1}^n \lambda'_{IW_\alpha}, \mathcal{A}_{OW_\alpha} = \mathcal{A}'_{IW_\alpha}.$$
5. For each bool gate ($IW_\alpha, IW_\beta, OW_\gamma, NAND$), each party performs a series of \mathcal{F}_{bitOT} and $\mathcal{F}_{stringOT}$. Then they construct the garbled tables for the gate. The garbled table GG_γ^i constructed by P_i ($i \geq 2$) is as follows (P_i sends GG_1^i, \dots, GG_g^i to P_1):

$$H(L'_{IW_\alpha,0}, L'_{IW_\beta,0}, \gamma, 00) \oplus (\mathcal{A}_{OW_\gamma,00}^i \oplus (L_{OW_\gamma, \mathcal{A}_{OW_\gamma,00}}^i)^{i \neq 1})$$

$$H(L'_{IW_\alpha,0}, L'_{IW_\beta,1}, \gamma, 01) \oplus (\mathcal{A}_{OW_\gamma,01}^i \oplus (L_{OW_\gamma, \mathcal{A}_{OW_\gamma,01}}^i)^{i \neq 1})$$

$$H(L'_{IW_\alpha,1}, L'_{IW_\beta,0}, \gamma, 10) \oplus (\mathcal{A}_{OW_\gamma,10}^i \oplus (L_{OW_\gamma, \mathcal{A}_{OW_\gamma,10}}^i)^{i \neq 1})$$

$$H(L'_{IW_\alpha,1}, L'_{IW_\beta,1}, \gamma, 11) \oplus (\mathcal{A}_{OW_\gamma,11}^i \oplus (L_{OW_\gamma, \mathcal{A}_{OW_\gamma,11}}^i)^{i \neq 1})$$

P_1 holds the secret share of garbled table as follows:

$$(\mathcal{A}_{OW_\gamma,00}^1 \oplus (L_{OW_\gamma, \mathcal{A}_{OW_\gamma,00}}^1)^{i \neq 1})$$

$$(\mathcal{A}_{OW_\gamma,01}^1 \oplus (L_{OW_\gamma, \mathcal{A}_{OW_\gamma,01}}^1)^{i \neq 1})$$

$$(\mathcal{A}_{OW_\gamma,10}^1 \oplus (L_{OW_\gamma, \mathcal{A}_{OW_\gamma,10}}^1)^{i \neq 1})$$

$$(\mathcal{A}_{OW_\gamma,11}^1 \oplus (L_{OW_\gamma, \mathcal{A}_{OW_\gamma,11}}^1)^{i \neq 1})$$

Online:

6. For the input wire I_j of the circuit C_f , P_i ($i \neq j$) sends the secret share $\lambda_{I_j}^i$ of the mask bit λ_{I_j} to P_j . P_j computes the mask bit $\lambda_{I_j} = \bigoplus_{i=1}^n \lambda_{I_j}^i$ and masked value $A_{I_j} = \lambda_{I_j} \oplus x_{I_j}$ where x_{I_j} is the real input of P_j . P_j sends A_{I_j} to all other parties.
7. P_i ($i \geq 2$) holds the masked value A_I corresponding to all input wires I of the C_f . Then P_i sends the key labels L_{I, A_I}^i to P_1 . Finally P_1 holds the masked value A_I and the corresponding key labels $\{L_{I, A_I}^i\}_{i \geq 2}$.
8. For the gate ($IW_\alpha, IW_\beta, OW_\gamma, NAND$), since P_1 holds π , P_1 knows the "outgoing wire" that connects to IW_α is OW_α and connects to IW_β is OW_β . P_1 holds $\{\mathcal{A}_{OW_\alpha, i}^1\}_{i \neq 1}$ and $\{\mathcal{A}_{OW_\beta, i}^1\}_{i \neq 1}$. From step 4 we can know $\mathcal{A}_\alpha = \mathcal{A}_{OW_\alpha} = \mathcal{A}'_{IW_\alpha}$, $\mathcal{A}_\beta = \mathcal{A}_{OW_\beta} = \mathcal{A}'_{IW_\beta}$. P_1 computes:

$$L'_{IW_\alpha, \mathcal{A}_\alpha} = L_{OW_\alpha, \mathcal{A}_\alpha}^1 \oplus T_{IW_\alpha}^1, L'_{IW_\beta, \mathcal{A}_\beta} = L_{OW_\beta, \mathcal{A}_\beta}^1 \oplus T_{IW_\beta}^1$$
9. P_1 performs the conversion in step 8 and evaluates each *NAND* gate according to the topological order of the circuit. P_1 computes $l = 2\mathcal{A}_\alpha + \mathcal{A}_\beta$ and the following ($i \geq 2$):

$$(A_{I, l}^i \oplus (L_{OW_\gamma, \mathcal{A}_{I, l}}^i)^{i \neq 1}) \oplus (L_{OW_\gamma, \mathcal{A}_{I, l}}^i)^{i \neq 1} = H(L'_{IW_\alpha, \mathcal{A}_\alpha}, L'_{IW_\beta, \mathcal{A}_\beta}, \gamma, l) \oplus GG_\gamma^i.$$

P_1 computes $\mathcal{A}_\gamma = \bigoplus_{j=1}^n \mathcal{A}_{\gamma, l}^j$, $L_{OW_\gamma, \mathcal{A}_\gamma} = \bigoplus_{j=1}^n (L_{OW_\gamma, \mathcal{A}_{\gamma, l}}^j)^j$.

Output:

10. For the output wire O , P_1 holds the masked value \mathcal{A}_O . P_i ($i \geq 2$) sends the secret share λ_O^i of λ_O on the output wire O to P_1 . P_1 computes $\lambda_O = \bigoplus_{j=1}^n \lambda_O^j$ and the result $y = \mathcal{A}_O \oplus \lambda_O$.

Fig. 6. The PFE protocol

And P_i holds the secret shares below:

$$\begin{aligned}
(L_{OW_\gamma, A_{\gamma,00}}^i)^i &= L_{OW_\gamma,0}^i \oplus (A_{\gamma,00} \Delta_i)^i \\
(L_{OW_\gamma, A_{\gamma,01}}^i)^i &= L_{OW_\gamma,0}^i \oplus (A_{\gamma,00} \Delta_i)^i \oplus (\lambda'_{IW_\alpha} \Delta_i)^i \\
(L_{OW_\gamma, A_{\gamma,10}}^i)^i &= L_{OW_\gamma,0}^i \oplus (A_{\gamma,00} \Delta_i)^i \oplus (\lambda'_{IW_\beta} \Delta_i)^i \\
(L_{OW_\gamma, A_{\gamma,11}}^i)^1 &= L_{OW_\gamma,0}^i \oplus (A_{\gamma,00} \Delta_i)^i \oplus (\lambda'_{IW_\alpha} \Delta_i)^i \oplus (\lambda'_{IW_\beta} \Delta_i)^i \oplus \Delta_i
\end{aligned} \tag{2}$$

4.2 Optimize

This subsection is mainly divided into three parts. In the first part, we will briefly introduce how to use the half-gate technique in a 2-PFE protocol based on the standard garbled circuit. In the second part, we will introduce how to use the half-gate technique in our protocol. In the third part, we will introduce some other small optimizations.

Table 1. The half-gate form of the standard garbled circuit, where a, b, c represent the type of gate ($a = 0, b = 0, c = 1$ means $NAND$ gate $(\alpha, \beta, \gamma, NAND)$). x_α, x_β represent the truth values on the input wires α and β . $\lambda_\alpha, \lambda_\beta$ represent the permutation bits on the input wires α and β . Only garbler knows $\lambda_\alpha = lsb(L_{\alpha,0})$, $\lambda_\beta = lsb(L_{\beta,0})$. $L_{\alpha,0}, L_{\beta,0}$ correspond to the truth value 0. T_{G_c}, T_{E_c} represent the garbled table of the garbler half-gate and the evaluator half-gate, respectively. $W_{G_c,0}, W_{E_c,0}$ represent the key label of the truth value 0 corresponding to the output wire of the garbler half-gate and the evaluator half-gate. $L_{\gamma,0}, L_{\gamma,1}$ represent the key labels on the output wire γ .

Garbler half gate (λ_β known to the garbler)	Evaluator half gate ($\lambda_\beta \oplus x_\beta$ known to the evaluator)
Defines the half gate:	Defines the half gate
$f_G(x_\alpha, \lambda_\beta) := (a \oplus x_\alpha)(b \oplus \lambda_\beta) \oplus c$	$f_E(x_\alpha, x_\beta \oplus \lambda_\beta) := (a \oplus x_\alpha)(\lambda_\beta \oplus x_\beta)$
Computes:	Computes:
$T_{G_c} \leftarrow H(L_{\alpha,0}) \oplus H(L_{\alpha,1}) \oplus (\lambda_\beta \oplus b) \Delta$	$T_{E_c} \leftarrow H(L_{\beta,0}) \oplus H(L_{\beta,1}) \oplus L_{\alpha,a}$
$W_{G_c,0} \leftarrow H(L_{\alpha,\lambda_\alpha}) \oplus f_G(\lambda_\alpha, \lambda_\beta) \Delta$	$W_{E_c,0} \leftarrow H(L_{\beta,\lambda_\beta})$
$L_{\gamma,0} = W_{G_c,0} \oplus W_{E_c,0}$	
$L_{\gamma,1} = L_{\gamma,0} \oplus \Delta$	

Half-gate in 2-PFE. Table 1 shows the specific form of half-gate in the standard garbled circuit. For a gate $(\alpha, \beta, \gamma, NAND)$, the garbler P_2 needs to use the key labels $L_{\alpha,\lambda_\alpha}$ on $Wire_\alpha$ and L_{β,λ_β} on $Wire_\beta$ to construct $W_{G_c,0}, W_{E_c,0}$. Then P_2 computes the key labels $L_{\gamma,0}$ and $L_{\gamma,1}$ on $Wire_\gamma$. Thus, the key labels on $Wire_\gamma$ are related to the key labels of the input wires of the gate, and P_2 should first determine the key labels on $Wire_\alpha$ and $Wire_\beta$ before determining

the key label on $Wire_\gamma$. But the situation is different in the PFE protocol based on garbled circuit. For the gate $(IW_\alpha, IW_\beta, OW_\gamma, NAND)$, P_2 must first determine the key label on OW_γ for performing the OEP protocol. In order to use the half-gate technology in the PFE protocol, P_2 can still construct the half-gate garbled table in the form of Table 1, but the key labels $L_{\gamma,0}, L_{\gamma,1}$ are not the true key labels on OW_γ , when P_2 sends the half-gate garbled table to P_1 , an additional message needs to be attached. And after P_1 evaluates the half-gate garbled table, P_1 also needs to use the additional message which is sent by P_2 to convert the result to the true key label on the OW_γ . In this way, P_1 can correctly evaluate the entire garbled circuit. This is a special half gate in the PFE protocol that can reduce the communication overhead of k bits instead of $2k$ bits in each garbled table (k bits additional message will be sent by P_2). Note that there is no need to send an additional message for output gates of the circuit, because the output wires of the circuit are not the “outgoing wire” and don’t need to generate the key labels in advance. In [7], Bingöl and Biçer used the above idea to construct the half-gate in their 2-PFE protocol. The additional message sent by P_2 is $\psi_\gamma = W_{G_c,0} \oplus W_{E_c,0} \oplus L_{OW_\gamma,0}$. After evaluating the half-gate garbled table, if the result obtained by P_1 is $L_{\gamma,0}$ (corresponding to the truth value 0), then P_1 computes $L_{OW_\gamma,0} = L_{\gamma,0} \oplus \psi_\gamma$ (corresponding to the truth value 0). If the result obtained by P_1 is $L_{\gamma,1}$ (corresponding to the truth value 1), then P_1 computes $L_{OW_\gamma,1} = L_{\gamma,1} \oplus \psi_\gamma$ (corresponding to the truth value 1). So the conversion guarantees the correctness of the result.

Table 2. The half-gate form of standard garbled circuit. Main difference from Table 1 is that $L_{\alpha,0}, L_{\beta,0}, L_{\gamma,0}$ represent the masked value “0” on the corresponding wires instead of the truth value 0. λ represents the mask bit on the wire, but it is still determined by the garbler in the standard garbled circuit. $\lambda_{\gamma_1}, \lambda_{\gamma_2}$ represent the mask bits on the output wires of the garbler half-gate and the evaluator half-gate, respectively.

Garbler half gate	Evaluator half gate
Computes:	Computes:
$T_{G_c} \leftarrow H(L_{\alpha,0}) \oplus H(L_{\alpha,1}) \oplus \lambda_\beta \Delta$	$T_{E_c} \leftarrow H(L_{\beta,0}) \oplus H(L_{\beta,1}) \oplus L_{\alpha,0} \oplus \lambda_\alpha \Delta$
$W_{G_c,0} \leftarrow H(L_{\alpha,0}) \oplus (\lambda_\alpha \lambda_\beta \oplus \lambda_{\gamma_1} \oplus 1) \Delta$	$W_{E_c,0} \leftarrow H(L_{\beta,0}) \oplus \lambda_{\gamma_2} \Delta$
$\lambda_\gamma = \lambda_{\gamma_1} \oplus \lambda_{\gamma_2}$	
$L_{\gamma,0} = W_{G_c,0} \oplus W_{E_c,0} = H(L_{\alpha,0}) \oplus H(L_{\beta,0}) \oplus (\lambda_\alpha \lambda_\beta \oplus \lambda_\gamma \oplus 1) \Delta$	

Half-gate in Our Protocol. The half-gate construction in the 2-PFE protocol can not be directly applied to our constant-round multiparty PFE protocol. To understand the difference more clearly, we first use the symbolic representation in our protocol to describe the half-gate construction in Table 1, the specific form is shown in Table 2 (the gate type is $NAND$). In our PFE protocol, for a gate $(IW_\alpha, IW_\beta, OW_\gamma, NAND)$, P_i constructs the following garbled table, $i \geq 2$:

$$\Lambda_\alpha, \Lambda_\beta \in \{0, 1\}$$

$$H(L_{IW_\alpha, \Lambda_\alpha}^{i'}, L_{IW_\beta, \Lambda_\beta}^{i'}, \gamma, \Lambda_\alpha \Lambda_\beta) \oplus (\Lambda_{\gamma, \Lambda_\alpha \Lambda_\beta}^i, (L_{\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^i)^i, \{(L_{\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^j)^i\}_{j \neq i, 1})$$

At first, the garbled table constructed by P_i is not a standard garbled table format. The encrypted content of each row is a triple instead of a key label on the output wire of the gate. Recently, Yang et al. proposed the idea of Partial Half-Gate in their malicious SFE protocol [32]. Similar to their trick, we first split each row in the garbled table into two parts:

$$A_{\Lambda_\alpha \Lambda_\beta}^i := H(L_{IW_\alpha, \Lambda_\alpha}^{i'}, L_{IW_\beta, \Lambda_\beta}^{i'}, \gamma, \Lambda_\alpha \Lambda_\beta) \oplus (L_{\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^i)^i,$$

$$B_{\Lambda_\alpha \Lambda_\beta}^i := H'(L_{IW_\alpha, \Lambda_\alpha}^{i'}, L_{IW_\beta, \Lambda_\beta}^{i'}, \gamma, \Lambda_\alpha \Lambda_\beta) \oplus (\Lambda_{\gamma, \Lambda_\alpha \Lambda_\beta}^i, \{(L_{\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^j)^i\}_{j \neq i, 1}).$$

$A_{\Lambda_\alpha \Lambda_\beta}^i$ conforms to the form of the garbled table in the standard garbled circuit, so we can try to construct the half-gate on it. Since λ'_{IW_α} , λ'_{IW_β} , λ_{OW_γ} are no longer held by P_2 but shared between n parties, each garbler P_i can not construct T_{G_c} , T_{E_c} , $L_{\gamma, 0}$ in Table 2. But P_i can use the secret shares generated by performing the functionalities \mathcal{F}_{bitOT} and $\mathcal{F}_{stringOT}$ to construct the following half-gate garbled table:

$$T_{G_c}^i := H(L_{IW_\alpha, 0}^{i'}) \oplus H(L_{IW_\alpha, 1}^{i'}) \oplus (\lambda'_{IW_\beta} \Delta_i)^i,$$

$$T_{E_c}^i := H(L_{IW_\beta, 0}^{i'}) \oplus H(L_{IW_\beta, 1}^{i'}) \oplus L_{IW_\alpha}^{i'} \oplus (\lambda'_{IW_\alpha} \Delta_i)^i,$$

$$L_{\gamma, 0}^i := H(L_{IW_\alpha, 0}^{i'}) \oplus H(L_{IW_\beta, 0}^{i'}) \oplus ((\lambda'_{IW_\alpha} \lambda'_{IW_\beta} \oplus \lambda_{OW_\gamma} \oplus 1) \Delta_i)^i.$$

P_i ($i \geq 2$) sends $T_{G_c}^i, T_{E_c}^i, \{B_{\Lambda_\alpha \Lambda_\beta}^i\}_{\Lambda_\alpha, \Lambda_\beta \in \{0, 1\}}, \psi_\gamma^i = L_{\gamma, 0}^i \oplus L_{OW_\gamma, 0}^i$ to P_1 . When evaluating the gate $(IW_\alpha, IW_\beta, OW_\gamma, NAND)$, P_1 has the masked values $\Lambda_\alpha, \Lambda_\beta$ and the corresponding key labels $\{L_{IW_\alpha, \Lambda_\alpha}^{i'}, L_{IW_\beta, \Lambda_\beta}^{i'}\}_{i \neq 1}$. For $i \geq 2$, P_1 first evaluates the half-gate part (we refer readers to read [34] in detail to learn the evaluation of half-gate):

$$Eval(\Lambda_\alpha, \Lambda_\beta, L_{IW_\alpha, \Lambda_\alpha}^{i'}, L_{IW_\beta, \Lambda_\beta}^{i'})$$

$$= H(L_{IW_\alpha, \Lambda_\alpha}^{i'}) \oplus H(L_{IW_\beta, \Lambda_\beta}^{i'}) \oplus \Lambda_\alpha T_{G_c}^i \oplus \Lambda_\beta (T_{E_c}^i \oplus L_{IW_\alpha, \Lambda_\alpha}^{i'})$$

$$= H(L_{IW_\alpha, 0}^{i'}) \oplus H(L_{IW_\beta, 0}^{i'}) \oplus \Lambda_\alpha (\lambda'_{IW_\beta} \Delta_i)^i \oplus \Lambda_\beta (\lambda'_{IW_\alpha} \Delta_i)^i \oplus \Lambda_\alpha \Lambda_\beta \Delta_i.$$

P_1 evaluates the $\{B_{\Lambda_\alpha \Lambda_\beta}^i\}_{\Lambda_\alpha, \Lambda_\beta \in \{0, 1\}}$ part:

$$(\Lambda_{\gamma, \Lambda_\alpha \Lambda_\beta}^i, \{(L_{\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^j)^i\}_{j \neq i, 1}) = H'(L_{IW_\alpha, \Lambda_\alpha}^{i'}, L_{IW_\beta, \Lambda_\beta}^{i'}, \Lambda_\alpha, \Lambda_\beta) \oplus B_{\Lambda_\alpha, \Lambda_\beta}^i.$$

P_1 holds $\Lambda_{\gamma, \Lambda_\alpha \Lambda_\beta}^1$ and $\{(L_{\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^j)^1\}_{j \neq 1}$. After evaluating the two portions, for $i \geq 2$, P_1 combines the results:

$$\begin{aligned}
\Lambda_\gamma &= \Lambda_{\gamma, \Lambda_\alpha \Lambda_\beta} = \bigoplus_{j=1}^n \Lambda_{\gamma, \Lambda_\alpha \Lambda_\beta}^j \\
\text{Eval}(\Lambda_\alpha, \Lambda_\beta, L'_{IW_\alpha, \Lambda_\alpha}, L'_{IW_\beta, \Lambda_\beta}) &\oplus \left(\bigoplus_{j \neq i}^n (L_{\gamma, \Lambda_\gamma}^i)^j \right) \\
&= H(L'_{IW_\alpha, 0}) \oplus H(L'_{IW_\beta, 0}) \oplus ((\lambda'_{IW_\alpha} \lambda'_{IW_\beta} \oplus \lambda_{OW_\gamma} \oplus 1) \Delta_i)^i \oplus \Lambda_\gamma \Delta_i \\
&= L_{\gamma, 0}^i \oplus \Lambda_\gamma \Delta_i \\
&= L_{\gamma, \Lambda_\gamma}^i.
\end{aligned} \tag{3}$$

Correctness. We verify the correctness of Eq. (3). In Eq. (1), we describe the secret share of $\{L_{OW_\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^i\}_{\Lambda_\alpha, \Lambda_\beta \in \{0, 1\}}$ held by P_j ($j \neq i$). We can simplify the four equations in Eq. (1):

$$\begin{aligned}
\Lambda_\alpha, \Lambda_\beta \in \{0, 1\}, (L_{OW_\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^i)^j &= (\Lambda_{\gamma, 00} \Delta_i)^j \oplus \Lambda_\alpha (\lambda'_{IW_\beta} \Delta_i)^j \oplus \Lambda_\beta (\lambda'_{IW_\alpha} \Delta_i)^j, \\
\Lambda_{\gamma, 00} &= \lambda'_{IW_\alpha} \lambda'_{IW_\beta} \oplus \lambda_{OW_\gamma} \oplus 1.
\end{aligned}$$

To use half-gate technique, we need to replace $L_{OW_\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^i$ with $L_{\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^i$. We set $S_i := \lambda'_{IW_\alpha} \Delta_i$, $R_i := \lambda'_{IW_\beta} \Delta_i$, $V_i := \Lambda_{\gamma, 00} \Delta_i$. The final result is as follows:

$$(L_{\gamma, \Lambda_\gamma, \Lambda_\alpha \Lambda_\beta}^i)^j = V_i^j \oplus \Lambda_\alpha R_i^j \oplus \Lambda_\beta S_i^j.$$

For $\Lambda_\gamma \Delta_i$, there is the following equation:

$$\begin{aligned}
&\Lambda_\gamma \Delta_i \\
&= ((\Lambda_\alpha \oplus \lambda'_{IW_\alpha})(\Lambda_\beta \oplus \lambda'_{IW_\beta}) \oplus \lambda_{OW_\gamma} \oplus 1) \Delta_i \\
&= \Lambda_\alpha \Lambda_\beta \Delta_i \oplus \Lambda_\alpha \lambda'_{IW_\beta} \Delta_i \oplus \Lambda_\beta \lambda'_{IW_\alpha} \Delta_i \oplus \Lambda_{\gamma, 00} \Delta_i \\
&= \Lambda_\alpha \Lambda_\beta \Delta_i \oplus \Lambda_\alpha R_i^i \oplus \Lambda_\beta S_i^i \oplus V_i^i \oplus \bigoplus_{j \neq i}^n (\Lambda_\alpha R_i^j \oplus \Lambda_\beta S_i^j \oplus V_i^j) \\
&= \Lambda_\alpha \Lambda_\beta \Delta_i \oplus \Lambda_\alpha R_i^i \oplus \Lambda_\beta S_i^i \oplus V_i^i \oplus \bigoplus_{j \neq i}^n ((L_{\gamma, \Lambda_\gamma}^i)^j).
\end{aligned}$$

For $\bigoplus_{j \neq i}^n ((L_{\gamma, \Lambda_\gamma}^i)^j)$, there is the following equation:

$$\bigoplus_{j \neq i}^n ((L_{\gamma, \Lambda_\gamma}^i)^j) = \Lambda_\gamma \Delta_i \oplus \Lambda_\alpha \Lambda_\beta \Delta_i \oplus \Lambda_\alpha R_i^i \oplus \Lambda_\beta S_i^i \oplus V_i^i. \tag{4}$$

We can verify Eq. (3) by replacing $\bigoplus_{j \neq i}^n ((L_{\gamma, \Lambda_\gamma}^i)^j)$ in Eq. (3) with Eq. (4). Finally, P_1 learns the key label on “outgoing wire” OW_γ by computing $L_{OW_\gamma, \Lambda_\gamma}^i = L_{\gamma, \Lambda_\gamma}^i \oplus \psi_\gamma^i$.

Other Optimizations. For the mask bit on the input wire I_i , P_i can randomly select the secret share $\lambda_{I_i}^i$ and P_j ($j \neq i$) can set $\lambda_{I_i}^j = 0$, so P_i can generate masked value A_{I_i} locally, thus saving 1 round of communication. In addition, the garblers P_i ($i \geq 2$) can make $lsb(\Delta_i) = 1$, so that the parties do not need to encrypt the secret share of the masked value in the garbled table, but send an extra bit $lsb(L_{\gamma, 0}^i)$ (see Fig. 7 for detailed description). A similar idea is used in [17, 31].

In Fig. 7, we describe the optimized protocol in detail. The final protocol is also divided into the preprocessing stage and the online stage. Our protocol \prod_{n-PFE} is secure for any corrupted parties under the semi-honest security model. In Appendix B, we prove the following theorem:

Theorem 1. *If H is modeled as a random oracle, the n -PFE protocol in Fig. 7 is secure in the $(\mathcal{F}_{n-OSN}, \mathcal{F}_{bitOT}, \mathcal{F}_{stringOT})$ -hybrid model against an semi-honest adversary corrupting up to $n - 1$ parties.*

4.3 HE-OEP

In addition to using the OSN protocol to implement OEP (OSN-OEP), we can also use singly homomorphic encryption [10, 25] to implement OEP (HE-OEP). The main difference is step 4 of Fig. 7. In the construction based on singly homomorphic encryption, P_i ($i \geq 2$) needs to generate a public key pk_i , and then uses pk_i to encrypt the inputs of the OEP protocol. For key labels, P_i sends the following messages to P_1 :

$$pk_i, Enc_{pk_i}(L_{1,0}^i), \dots, Enc_{pk_i}(L_{M,0}^i).$$

P_1 uses the information π to perform *extended permutation* on the messages sent by P_i . After performing *extended permutation*, P_1 uses the property of singly homomorphic encryption to blind the results and sends the blinded results to P_i :

$$Enc_{pk_i}(L_{\pi^{-1}(1),0}^i + T_1^i), \dots, Enc_{pk_i}(L_{\pi^{-1}(N),0}^i + T_N^i).$$

Protocol \prod_{n-PFE}

Input: It is same as Fig. 6

Preprocessing:

1. It is same as Fig. 6.
2. It is same as Fig. 6, except that for the input wire I_j of P_j ($j \in [n]$), P_j randomly selects a secret share $\lambda_{I_j}^j$ and P_i ($i \neq j$) sets $\lambda_{I_j}^i = 0$.
3. It is same as Fig. 6, except that P_i ($i \geq 2$) sets $lsb(\Delta_i) = 1$ and defer to the step 5 to generate key label $L_{O,0}^i$.
4. It is same as Fig. 6.
5. For each bool gate ($IW_\alpha, IW_\beta, OW_\gamma, NAND$), each party performs a series of \mathcal{F}_{bitOT} and $\mathcal{F}_{stringOT}$. Then they construct the garbled table for the gate. P_i ($i \geq 2$) computes (for the output gate of C_f , no need to construct ψ_γ^i):

$$T_{G_c}^i := H(L'_{IW_\alpha,0}) \oplus H(L'_{IW_\alpha,1}) \oplus (\lambda'_{IW_\beta} \Delta_i)^i,$$

$$T_{E_c}^i := H(L'_{IW_\beta,0}) \oplus H(L'_{IW_\beta,1}) \oplus L'_{IW_\alpha,0} \oplus (\lambda'_{IW_\alpha} \Delta_i)^i,$$

$$L_{\gamma,0}^i := H(L'_{IW_\alpha,0}) \oplus H(L'_{IW_\beta,0}) \oplus ((\lambda'_{IW_\alpha} \lambda'_{IW_\beta} \oplus \lambda_{OW_\gamma} \oplus 1) \Delta_i)^i,$$

$$\psi_\gamma^i = L_{\gamma,0}^i \oplus L_{OW_\gamma,0}^i,$$

$$A_\alpha, A_\beta \in \{0, 1\},$$

$$B_{A_\alpha A_\beta}^i := H(L'_{IW_\alpha, A_\alpha}, L'_{IW_\beta, A_\beta}, \gamma, A_\alpha A_\beta) \oplus (\{(L_{\gamma, A_\gamma, A_\alpha A_\beta}^j)^i\}_{j \neq i, 1}).$$

P_i sends $T_{G_c}^i, T_{E_c}^i, \{B_{A_\alpha A_\beta}^i\}_{A_\alpha, A_\beta \in \{0,1\}}, \psi_\gamma^i, b_\gamma^i = lsb(L_{\gamma,0}^i)$ to P_1 .

P_1 holds the secret share as follows:

$$\{(L_{\gamma, A_\gamma, 00}^i)^1\}_{i \neq 1} \quad \{(L_{\gamma, A_\gamma, 01}^i)^1\}_{i \neq 1} \quad \{(L_{\gamma, A_\gamma, 10}^i)^1\}_{i \neq 1} \quad \{(L_{\gamma, A_\gamma, 11}^i)^1\}_{i \neq 1}.$$

Online:

6. For the input wire I_j , P_j ($j \in [n]$) directly computes $A_{I_j} = x_{I_j} \oplus \lambda_{I_j}^j$, and sends the masked value A_{I_j} to other parties.
7. It is same as Fig. 6.
8. It is same as Fig. 6.
9. P_1 performs the conversion in step 8 and evaluates each $NAND$ gate according to the topological order of the circuit C_f . P_1 computes $l = 2A_\alpha + A_\beta$, for $i \geq 2$ P_1 computes:

$$\{(L_{\gamma, A_\gamma}^j)^i\}_{j \neq i, 1} = H(L'_{IW_\alpha, A_\alpha}, L'_{IW_\beta, A_\beta}, \gamma, l) \oplus B_l^i,$$

$$L_{\gamma, A_\gamma}^i = H(L'_{IW_\alpha, A_\alpha}) \oplus H(L'_{IW_\beta, A_\beta}) \oplus A_\alpha T_{G_c}^i \oplus A_\beta (T_{E_c}^i \oplus L'_{IW_\alpha, A_\alpha}) \oplus (\bigoplus_{j \neq i}^n (L_{\gamma, A_\gamma}^j)^i),$$

$$A_\gamma = lsb(L_{\gamma, A_\gamma}^i) \oplus b_\gamma^i,$$

$$L_{OW_\gamma, A_\gamma}^i = L_{\gamma, A_\gamma}^i \oplus \psi_\gamma^i.$$

Output:

10. It is same as Fig. 6.

Fig. 7. The optimized PFE protocol

P_i decrypts the ciphertexts sent by P_1 to learn $(L_{1,0}^{i'}, \dots, L_{N,0}^{i'})$. This process completes the OEP. For λ_j^i ($j \in [M]$), the process is same as $L_{j,0}^i$. In fact, $L_{j,0}^i$ and λ_j^i can be concatenated into one element as they are being encrypted, hence avoiding additional costs. Our protocol based on singly homomorphic encryption can achieve linear communication and computation complexities. In the OEP stage, HE-OEP only costs 2 rounds of communication, compared with the OSN-OEP costs 3 rounds of communication.

5 Efficiency Analysis

In this section, we will analyze the concrete efficiency of the optimized protocol in Fig. 7. We use $OT(P_i, P_j)$ to represent the OT protocol where P_i is the sender and P_j is the receiver, and all OT protocols in our protocol are optimized by using OT extension [2, 15, 19]. The authors of [23] construct a n -PFE protocol based on their framework which needs to use $O(n^2g + g \log g)$ invocations of OT ($O(g \log g)$ for offline and $O(n^2g)$ for online) and the round complexity is $O(g)$. At online stage, their protocol needs to use 1-out-of-4 OT protocols, and the OT protocols can not be optimized by using OT extension because they can not execute in parallel. Our OSN-based protocol has a similar asymptotic complexity, but actually the communication overhead is higher because the garbled tables need to be sent. We also compare the 2-PFE protocols that rely on symmetric operations with our OSN-based n -PFE protocol in Table 3. Table 4 shows the comparison of specific communication overhead, and the overhead of our protocol is about $1.3\times$ that of [23] and $2.3\times$ that of [7] for $n = 3$.

5.1 Communication Complexity

We omit the communication overhead of exchanging masked value among n parties and that of P_i ($i \geq 2$) sending the key labels of the input wires to P_1 . We also omit the communication overhead of the seed OT in the OT extension.

Table 3. Complexities comparison with the 2-PFE protocols that basically only rely on symmetric operations. We assume that P_i ($i \geq 2$) needs to send ψ_γ^i at each gate and omit the overhead of bit b_γ^i in our protocol.

Protocols	Communication	Computation	Rounds
MS13-2Party [23]	$(10N \log(N) + 4N + 5)k$	$(4N \log(N) + 2.5N + 2)$ Sym.+ $O(k)$ Asym.	6
BBKL19-2Party [7]	$(6N \log(N) + 0.5N + 3)k$	$(4N \log(N) + N + 2)$ Sym.+ $O(k)$ Asym.	6
Ours-OSN-nParty	$((6N \log(N) - N + 3)k + 4N \log(N) + 2 + 2gn + 14gk(n-1))(n-1)$	$(4N \log(N) + 2 + g + 8gn)(n-1)$ Sym. + $O(k)$ Asym.	9

Table 4. Comparison of specific communication overhead, $k = 128$ bits

Protocols	Num of gates					
	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
MS13-2Party [23]	3.56 MB	16.75 MB	77.00 MB	348.00 MB	1.52 GB	6.69 GB
BBKL19-2Party [7]	2.08 MB	9.81 MB	45.25 MB	205.00 MB	916.00 MB	3.95 GB
Ours-OSN ($n = 3$)	4.96 MB	22.86 MB	103.49 MB	462.22 MB	1.99 GB	8.73 GB

\mathcal{F}_{n-OSN} stage. $i \geq 2$, $P_i \rightarrow P_1 : N(k+1)$ bits. The inputs of P_i in the functionality \mathcal{F}_{n-OSN} contain N bit strings, and each string is of length $k+1$. For $OT(P_i, P_1)$, $P_1 \rightarrow P_i : (2N \log(N) - N + 1)k$ bits. $P_i \rightarrow P_1 : (4N \log(N) - 2N + 2)(k+1)$ bits. In the functionality \mathcal{F}_{n-OSN} , the total OT overhead is $2N \log(N) - N + 1$ bits. $P_1 \rightarrow P_i : N(k+1)$ bits. P_1 sends the final output of the \mathcal{F}_{n-OSN} to P_i . The total communication overhead of our protocol is $((6N \log(N) - N + 3)k + 4N \log(N) + 2)(n-1)$ bits. If we use the singly homomorphic encryption to implement OEP, the communication complexity of this stage is O/ng .

\mathcal{F}_{bitOT} and $\mathcal{F}_{stringOT}$ stage. In this stage, our protocol will cost $gn(n-1)$ \mathcal{F}_{bitOT} and $3g(n-1)^2$ $\mathcal{F}_{stringOT}$. In addition to the OT extension, for $\mathcal{F}_{stringOT}$ we can also use a variant of OT (i.e., *correlated OT* [2, 32]) for higher efficiency.

Garbled table. $i \geq 2$, $P_i \rightarrow P_1 : (4nk - 5k + 1)(g - O) + (4nk - 6k + 1)O$ bits. For each non-output gate, P_i sends $T_{G_c}^i, T_{E_c}^i, \psi_\gamma^i, b_\gamma^i = lsb(L_{\gamma,0}^i), \{B_{\Lambda_\alpha \Lambda_\beta}^i\}_{\Lambda_\alpha, \Lambda_\beta \in \{0,1\}}$ to P_1 (in fact, only one garbler needs to send b_γ). For each output gate, ψ_γ^i does not need to be sent by P_i .

Note that the main communication overhead occurs in the \mathcal{F}_{n-OSN} stage. When the network bandwidth is the bottleneck, we can use singly homomorphic encryption to implement OEP which will reduce the communication overhead, but it needs to cost O/ng asymmetric operations, the running time is not necessarily faster than the construction based on OSN protocol [14, 23].

5.2 Computation Complexity

For the OT extension, $O(k)$ asymmetric operations need to be performed, and each OT in the OT extension costs 2 symmetric operations.

\mathcal{F}_{n-OSN} stage: $(4N \log(N) - 2N + 2)(n-1)$ symmetric operations. For each 2-OSN, this stage needs to cost $2N \log(N) - N + 1$ OTs. Note that if we implement the OEP by using singly homomorphic encryption, this stage will cost O/ng asymmetric operations.

\mathcal{F}_{bitOT} and $\mathcal{F}_{stringOT}$ stage: $2gn(n-1) + 6g(n-1)^2$ symmetric operations. In this stage, our protocol will cost $gn(n-1)$ \mathcal{F}_{bitOT} and $3g(n-1)^2$ $\mathcal{F}_{stringOT}$.

Garbled table: $8g(n-1)$ symmetric operations. We can know from $T_{G_c}^i, T_{E_c}^i, L_{\gamma,0}^i, \{B_{\Lambda_\alpha \Lambda_\beta}^i\}_{\Lambda_\alpha, \Lambda_\beta \in \{0,1\}}$ that P_i needs to cost 8 symmetric operations at each gate.

Computation. P_1 : $3(n-1)g$ symmetric operations. P_1 needs to evaluate $n-1$ garbled tables when evaluating each gate, and the evaluation of each garbled table needs to cost 3 symmetric operations.

5.3 Round Complexity

We will analyze the total number of rounds in the preprocessing stage and the online stage separately. It cost 2 rounds of communication in the OT extension.

Round(pre): 6 or 7. The functionality \mathcal{F}_{n-OSN} costs 3 rounds of communication. But if we implement the OEP by using singly homomorphic encryption, it just costs 2 rounds of communication. A series of \mathcal{F}_{bitOT} can run in parallel, and a series of $\mathcal{F}_{stringOT}$ also can run in parallel. But \mathcal{F}_{bitOT} can not run in parallel with $\mathcal{F}_{stringOT}$, because our n -PFE protocol needs to use the result of \mathcal{F}_{bitOT} when running the functionality $\mathcal{F}_{stringOT}$. So \mathcal{F}_{bitOT} and $\mathcal{F}_{stringOT}$ need cost 4 rounds of communication in total.

Round(online): 2. It takes 1 round of communication to exchange masked value between n parties and P_i ($i \geq 2$) sends the garbled tables and the secret share λ_O^i to P_1 . It takes 1 round of communication for P_i ($i \geq 2$) to send the key labels L_{I,A_I}^i to P_1 .

So our protocols cost 8 or 9 rounds of communication in total.

Acknowledgement. Xiangxue Li is supported by National Natural Science Foundation of China (61971192), Shanghai Municipal Education Commission (2021-01-07-00-08-E00101), and Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

A OSN Protocol

A.1 Switching Network and Permutation Network

A switching network SN is a set of interconnected switches that take N inputs and a set of selection bits, and output N values. Each switch in the network accepts two l -bit strings as inputs and outputs two l -bit strings. A switch with two selection bits is called a 2-switch (2-SW), and with one selection bit is called a 1-switch (1-SW). There are 4 exchange types in 2-switch. If the input of a 2-switch is (x_0, x_1) , and the selection bits are (s_0, s_1) , then its outputs are $y_0 = x_{s_0}$, $y_1 = x_{s_1}$. In the OSN protocol, only one 1-switch (two exchange types) needs to be used.

Definition 3 (Mapping for a Switching Network). *The mapping $\pi : \{1 \dots N\} \rightarrow \{1 \dots N\}$ corresponding to a switching network SN is defined such that $\pi(i) = j$ if and only if after evaluation of SN on the N inputs, the value of the input wire i is assigned to the output wire j .*

A permutation network is a special switching network, and its corresponding mapping is a permutation, so only one 1-switch needs to be used in the permutation network. When the input of a switch is (x_0, x_1) , the corresponding output has two types, i.e., (x_0, x_1) or (x_1, x_0) . An optimal permutation network is proposed in [28]. For any $N = 2^l$ inputs, there is a permutation network with $N \log(N) - N + 1$ switches and the depth is $2 \log(N) - 1$.

A.2 OSN

In [23], Mohassel and Sadeghian implemented a constant-round 2-OEP protocol using a combination of switching network and permutation network (they called the *oblivious evaluation of a switching network* (OSN) protocol). We need to utilize the OSN protocol, so here we introduce the implementation of their protocol. We also refer the readers to [23] for the details of the OSN. The OSN protocol is mainly composed of two components, (1) implements extended permutation by using SN and PN, and (2) oblivious evaluation of switching networks by using OT protocol.

Construct EP with SN and PN. For a switching network, N inputs can finally get N outputs, but for *extended permutation* $\pi : \{1 \dots M\} \rightarrow \{1 \dots N\}$, M inputs will finally get N outputs ($N \geq M$). To simulate an extended permutation using a switching network, in addition to M real inputs of EP, $N - M$ dummy values are required. In [23], the construction of the entire switching network is divided into three components: (1) dummy value placement, (2) replication, (3) permutation. Figure 8 shows a concrete example.

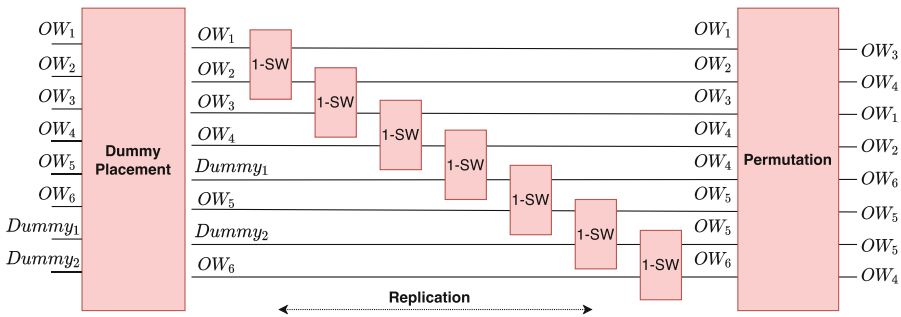


Fig. 8. A Switching Network for EP π

Dummy value placement component takes M real values and $N - M$ dummy values as inputs. For each real input that is mapped to k different outputs according to π , the component outputs the real value followed by $k - 1$ dummy values. The process is similar to permuting N inputs, so it can be implemented using a permutation network.

Replication component takes the outputs of the previous component as inputs. If the input is the real value, it will be output directly, and the dummy value will be replaced with the real value before it. Because only two exchange types are required, 1-switch can be used. For (x_0, x_1) , possible output is (x_0, x_1) or (x_0, x_0) . When outputting (x_0, x_1) , it means that both values are real values. When outputting (x_0, x_0) , it means that x_0 is the real value, and x_1 is the dummy value. This phase can be implemented using $N - 1$ switches.

Permutation component takes the outputs of the replication component as inputs, and N elements are placed in the final position according to the mapping relationship π . This component can also be implemented using a permutation network.

The first and third components can be implemented using a permutation network, so the number of switches required is $2(N \log(N) - N + 1)$. The second component requires $N - 1$ switches. Since 1-switch can be used in all three components, a total of $(2N \log(N) - N + 1)$ 1-switches are required to implement the switching network.

Oblivious Evaluation of Switching Networks (OSN). Next, we will show how 2-OEP can be achieved by computing the entire switching network. P_1 has a mapping π , so P_1 can get the selection bit of each switch in the switching network through π , and P_1 also has a blind vector \vec{t} , P_2 has the input vector \vec{x} . Finally, P_2 learns the output $(x_{\pi^{-1}(1)} \oplus t_1 \dots x_{\pi^{-1}(N)} \oplus t_N)$ of the switching network. In Fig. 9, we give two examples of 1-switch: Fig. 9(a) is mainly used for the dummy value component and permutation component, and Fig. 9(b) is used for the replication component. We take Fig. 9(a) as an example to explain, assuming that the input wires of a switch are w_i and w_j , the output wires are w_k and w_l , P_2 generates random values on all four wires r_i, r_j, r_k, r_l . P_1 has $(x_i \oplus r_i), (x_j \oplus r_j)$ and the selection bit s_0 of this switch. After evaluating the switch, P_1 learns y_1, y_2 , this can be implemented through the OT protocol where P_1 as the receiver inputs the selection bit s_0 and P_2 as the sender inputs the two values of the Γ column in Fig. 9(a). For example, when s_0 is 0, P_1 will get $(r_i \oplus r_k, r_j \oplus r_l)$, then P_1 uses $(x_i \oplus r_i)$ and $(r_i \oplus r_k)$ to perform XOR operation to get $(x_i \oplus r_k)$, and uses $(x_j \oplus r_j)$ and $(r_j \oplus r_l)$ to perform the XOR operation to get $(x_j \oplus r_l)$, which completes the evaluation of a switch. The evaluation process for the entire switching network is as follows. In the offline stage, P_2 generates a random value for each wire in the switching network, P_1 and P_2 execute a series of parallel 1-out-of-2 OT protocols. In the online stage, P_2 blinds its own input vector \vec{x} using the random values on the input wires of the switching network that are generated at the offline stage, and then sends the blinded result to P_1 . Now P_1 has the necessary information to evaluate the entire switching network, and uses the XOR operation to evaluate the entire switching network locally. Finally, P_1 uses the blinding vector \vec{t} to blind the output of the switching network and sends the blinded result to P_2 , P_2 unblinds the result using random values on the output wires of the switching network generated at the offline stage and learns the final result $(x_{\pi^{-1}(1)} \oplus t_1 \dots x_{\pi^{-1}(N)} \oplus t_N)$.

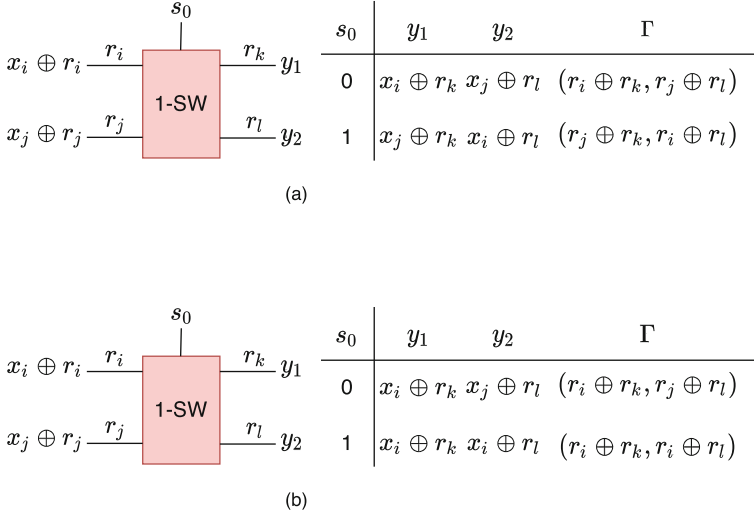


Fig. 9. 1-Switch

From the above description, we can know that the OSN protocol is constant-round. The total number of rounds is 3, because the OT protocol costs 2 rounds, and it also costs 1 round for P_1 to send the result to P_2 . Note that 1 round of communication in which P_2 sends the blinded vector \vec{x} to P_1 can be incorporated into the OT protocol. The evaluation of each switch needs to cost 1 OT, and the entire OSN protocol needs to cost $(2N \log(N) - N + 1)$ 1-out-of-2 OT protocols.

B Security Proof

Theorem 2. *If H is modeled as a random oracle, the n -PFE protocol in Fig. 7 is secure in the $(\mathcal{F}_{n-OSN}, \mathcal{F}_{bitOT}, \mathcal{F}_{stringOT})$ -hybrid model against a semi-honest adversary corrupting up to $n - 1$ parties.*

Proof. We will divide into two cases: $P_1 \in \mathcal{H}$, and $P_1 \in \mathcal{C}$ and $P_2 \in \mathcal{H}$. Note that the case of $P_1 \in \mathcal{C}$ and $P_i \in \mathcal{H}$ ($i \geq 3$) is similar to the second case.

$P_1 \in \mathcal{H}$. Let \mathcal{A} denote an adversary that corrupts $\{P_i\}_{i \in \mathcal{C}}$. We construct a simulator \mathcal{S} to simulate \mathcal{A} and play the role of $\{P_i\}_{i \in \mathcal{C}}$ in an ideal world involving an ideal functionality \mathcal{F}_{n-PFE} evaluating f . \mathcal{S} is defined as follows:

- 1-5 \mathcal{S} acts as honest $\{P_i\}_{i \in \mathcal{H}}$ and plays the functionalities \mathcal{F}_{n-OSN} , \mathcal{F}_{bitOT} and $\mathcal{F}_{stringOT}$, recording all values sent to and received from \mathcal{A} .
- 6 \mathcal{S} interacts with \mathcal{A} acting as honest $\{P_i\}_{i \in \mathcal{H}}$, using input $\{x_{I_i} := 0\}_{i \in \mathcal{H}}$ for input wire I_i , \mathcal{S} sends $\Lambda_{I_i} = x_{I_i} \oplus \lambda_{I_i}^i$ to \mathcal{A} . For each $i \in \mathcal{C}$ and each input wire I_i , \mathcal{S} receives Λ_{I_i} from \mathcal{A} and computes $x_{I_i} = \Lambda_{I_i} \oplus \lambda_{I_i}^i$.

- 7 \mathcal{S} interacts with \mathcal{A} acting as honest $\{P_i\}_{i \in \mathcal{H}}$, for each $i \in \mathcal{C}$ and all input wires I , \mathcal{S} receives L_{I, Λ_i}^i from \mathcal{A} .
- 8-10 \mathcal{S} interacts with \mathcal{A} acting as honest $\{P_i\}_{i \in \mathcal{H}}$. For each $i \in \mathcal{C}$, \mathcal{S} sends $(input, x_{I_i})$ on behalf of P_i to \mathcal{F}_{n-PFE} .

Next we will show that the joint distribution of the outputs of \mathcal{A} and honest $\{P_i\}_{i \in \mathcal{H}}$ in the real world is indistinguishable from the joint distribution of the outputs of \mathcal{S} and $\{P_i\}_{i \in \mathcal{H}}$ in the ideal world.

Hybrid1. This is the hybrid-world protocol. \mathcal{S} plays the role of honest $\{P_i\}_{i \in \mathcal{H}}$, using real input $\{x_{I_i}\}_{i \in \mathcal{H}}$ instead of making $\{x_{I_i} := 0\}_{i \in \mathcal{H}}$, and plays the role of \mathcal{F}_{n-OSN} , \mathcal{F}_{bitOT} , $\mathcal{F}_{stringOT}$.

Hybrid2. Same as **Hybrid1**, except in step 6, for each $i \in \mathcal{C}$, for each input wire I_i , \mathcal{S} receives Λ_{I_i} from \mathcal{A} , and computes $x_{I_i} = \Lambda_{I_i} \oplus \lambda_{I_i}^i$. Then \mathcal{S} sends $(input, x_{I_i})$ on behalf of P_i to \mathcal{F}_{n-PFE} . So P_1 outputs $\mathcal{F}(x_{I_1}, \dots, x_{I_n})$.

The distributions on the view of \mathcal{A} in **Hybrid1** and **Hybrid2** are identical.

P_1 will generate the same outputs in both **Hybrid1** and **Hybrid2** under the semi-honest security model.

Hybrid3. Same as **Hybrid2**, except in step 6, for each $i \in \mathcal{H}$, for each input wire I_i , \mathcal{S} uses $\{x_{I_i} := 0\}_{i \in \mathcal{H}}$ as input.

The distributions on the view of \mathcal{A} in **Hybrid2** and **Hybrid3** are identical, because $\{\lambda_{I_i}^i\}_{i \in \mathcal{H}}$ are uniform, so $\{\Lambda_{I_i}\}_{i \in \mathcal{H}}$ are uniform for \mathcal{A} .

Note that **Hybrid1** is the real-world execution and **Hybrid3** is the ideal-world execution. This completes the proof for honest P_1 .

$P_1 \in \mathcal{C}$, $P_2 \in \mathcal{H}$. Let \mathcal{A} denote an adversary that corrupts $\{P_i\}_{i \in \mathcal{C}}$. We construct a simulator \mathcal{S} to simulate \mathcal{A} and play the role of $\{P_i\}_{i \in \mathcal{C}}$ in an ideal world involving an ideal functionality \mathcal{F}_{n-PFE} evaluating f . \mathcal{S} is defined as follows:

- 1-4 \mathcal{S} acts as honest $\{P_i\}_{i \in \mathcal{H}}$ and plays the functionality \mathcal{F}_{n-OSN} , recording all values sent to and received from \mathcal{A} .
- 5 \mathcal{S} acts as honest $\{P_i\}_{i \in \mathcal{H}}$ and plays the functionalities \mathcal{F}_{bitOT} and $\mathcal{F}_{stringOT}$, and sends the garbled tables to \mathcal{A} .
- 6 \mathcal{S} interacts with \mathcal{A} acting as honest $\{P_i\}_{i \in \mathcal{H}}$, using input $\{x_{I_i} := 0\}_{i \in \mathcal{H}}$ for input wire I_i , \mathcal{S} sends $\Lambda_{I_i} = x_{I_i} \oplus \lambda_{I_i}^i$ to \mathcal{A} . For each $i \in \mathcal{C}$, for each input wire I_i , \mathcal{S} receives Λ_{I_i} from \mathcal{A} , and computes $x_{I_i} = \Lambda_{I_i} \oplus \lambda_{I_i}^i$.
- 7 \mathcal{S} interacts with \mathcal{A} acting as honest $\{P_i\}_{i \in \mathcal{H}}$, for each $i \in \mathcal{H}$, for all input wires I , \mathcal{S} sends L_{I, Λ_i}^i to \mathcal{A} .
- 8-10 \mathcal{S} interacts with \mathcal{A} acting as honest $\{P_i\}_{i \in \mathcal{H}}$. For each $i \in \mathcal{C}$, \mathcal{S} sends $(input, x_{I_i})$ computed in step 6 on behalf of P_i to \mathcal{F}_{n-PFE} .

Next we will show that the joint distribution of the outputs of \mathcal{A} and honest $\{P_i\}_{i \in \mathcal{H}}$ in the real world is indistinguishable from the joint distribution of the outputs of \mathcal{S} and $\{P_i\}_{i \in \mathcal{H}}$ in the ideal world.

Hybrid1. Same as the hybrid-word protocol. \mathcal{S} plays the roles of honest $\{P_i\}_{i \in \mathcal{H}}$, using real input $\{x_{I_i}\}_{i \in \mathcal{H}}$ instead of making $\{x_{I_i} := 0\}_{i \in \mathcal{H}}$, and plays the roles of \mathcal{F}_{n-OSN} , \mathcal{F}_{bitOT} , $\mathcal{F}_{stringOT}$.

Hybrid2. Same as **Hybrid1**, except in step 6, for each $i \in \mathcal{C}$, for each input wire I_i , \mathcal{S} receives Λ_{I_i} from \mathcal{A} , and computes $x_{I_i} = \Lambda_{I_i} \oplus \lambda_{I_i}^i$, and \mathcal{S} sends $(input, x_{I_i})$ on behalf of P_i to \mathcal{F}_{n-PFE} .

The distributions on the view of \mathcal{A} in **Hybrid1** and **Hybrid2** are identical.

Hybrid3. Same as **Hybrid2**, except in step 6, for each $i \in \mathcal{H}$, for each input wire I_i , \mathcal{S} uses $\{x_{I_i} := 0\}_{i \in \mathcal{H}}$ as input.

It follows from the security of garbling with H modeled as a random oracle and $\{\lambda_{I_i}^i\}_{i \in \mathcal{H}}$ are uniform that the distributions on the view of \mathcal{A} in **Hybrid3** and **Hybrid2** are identical.

Note that **Hybrid1** is the real-world execution and **Hybrid3** is the ideal-word execution. This completes the proof for corrupted P_1 .

References

1. Alhassan, M.Y., Günther, D., Kiss, Á., Schneider, T.: Efficient and scalable universal circuits. *J. Cryptol.* **33**(3), 1216–1271 (2020)
2. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: *CCS 2013*, pp. 535–548. ACM (2013)
3. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: Backes, M., Ning, P. (eds.) *ESORICS 2009*. LNCS, vol. 5789, pp. 424–439. Springer, Heidelberg (2009)
4. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols. In: *Proceedings of the Twenty-second Annual ACM STOC*, pp. 503–513 (1990)
5. Ben-Efraim, A., Lindell, Y., Omri, E.: Optimizing semi-honest secure multiparty computation for the internet. In: *CCS 2016*, pp. 578–590. ACM (2016)
6. Biçer, O., Bingöl, M.A., Kiraz, M.S., Levi, A.: Highly efficient and re-executable private function evaluation with linear complexity. *IEEE Transactions on Dependable and Secure Computing* (2020)
7. Bingöl, M.A., Biçer, O., Kiraz, M.S., Levi, A.: An efficient 2-party private function evaluation protocol based on half gates. *Comput. J.* **62**(4), 598–613 (2019)
8. Choi, S.G., Katz, J., Malozemoff, A.J., Zikas, V.: Efficient Three-Party Computation from Cut-and-Choose. In: Garay, J.A., Gennaro, R. (eds.) *CRYPTO 2014*. LNCS, vol. 8617, pp. 513–530. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_29
9. Demmler, D., Schneider, T., Zohner, M.: ABY - A framework for efficient mixed-protocol secure two-party computation. In: *NDSS 2015*. The Internet Society (2015)
10. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* **31**(4), 469–472 (1985)

11. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Proceedings of the 19th Annual ACM STOC, pp. 218–229. ACM (1987)
12. Günther, D., Kiss, Á., Schneider, T.: More Efficient Universal Circuit Constructions. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 443–470. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_16
13. Holz, M., Kiss, Á., Rathee, D., Schneider, T.: Linear-complexity private function evaluation is practical. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12309, pp. 401–420. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59013-0_20
14. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS 2012. The Internet Society (2012)
15. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9
16. Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_30
17. Katz, J., Ranellucci, S., Rosulek, M., Wang, X.: Optimizing authenticated garbling for faster secure two-party computation. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10993, pp. 365–391. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_13
18. Kiss, Á., Schneider, T.: Valiant’s universal circuit is practical. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 699–728. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_27
19. Kolesnikov, V., Kumaresan, R.: Improved OT extension for transferring short secrets. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 54–70. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_4
20. Lipmaa, H., Mohassel, P., Sadeghian, S.: Valiant’s universal circuit: improvements, implementation, and applications. Cryptology ePrint Archive, Paper 2016/017 (2016). <https://eprint.iacr.org/2016/017>
21. Liu, H., Yu, Yu., Zhao, S., Zhang, J., Liu, W., Hu, Z.: Pushing the limits of Valiant’s universal circuits: simpler, tighter and more compact. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 365–394. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_13
22. Liu, Y., Wang, Q., Yiu, S.: Making private function evaluation safer, faster, and simpler. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022. LNCS, vol. 13177, pp. 349–378. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-030-97121-2_13
23. Mohassel, P., Sadeghian, S.S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_33
24. Mohassel, P., Sadeghian, S.S., Smart, N.P.: Actively secure private function evaluation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 486–505. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_26
25. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16

26. Sadeghi, A., Schneider, T.: Generalized universal circuits for secure evaluation of private functions with application to data classification. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 336–353. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-642-00730-9_21
27. Valiant, L.G.: Universal circuits (preliminary report). In: Proceedings of the Eighth Annual ACM STOC, pp. 196–203 (1976)
28. Waksman, A.: A permutation network. *J. ACM (JACM)* **15**(1), 159–163 (1968)
29. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: CCS 2017, pp. 21–37. ACM (2017)
30. Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: CCS 2017, pp. 39–56. ACM (2017)
31. Yang, K., Wang, X., Zhang, J.: More efficient MPC from improved triple generation and authenticated garbling. In: CCS 2020, pp. 1627–1646. ACM (2020)
32. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: CCS 2020, pp. 1607–1626. ACM (2020)
33. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (SFCS1986), pp. 162–167. IEEE (1986)
34. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8



Predicate Private Set Intersection with Linear Complexity

Yaxi Yang¹, Jian Weng^{1(✉)}, Yufeng Yi¹, Changyu Dong^{2(✉)}, Leo Yu Zhang³,
and Jianying Zhou⁴

¹ Jinan University, Guangzhou, China
cryptjweng@gmail.com

² Guangzhou University, Guangzhou, China
changyu.dong@gmail.com

³ Griffith University, Nathan, QLD, Australia

⁴ Singapore University of Technology and Design, Singapore, Singapore

Abstract. Private Set Intersection (PSI) enables two parties to learn the intersection of their input sets without exposing other items that are not within the intersection. However, real-world applications often require more complex computations than just obtaining the intersection. In this paper, we consider the setting where each item in the input set has an associated payload, and the desired output is a subset of the intersection obtained by evaluating certain conditions over the payload. We call this new primitive Predicate Private Set Intersection (PPSI) and show its applicability in many different scenarios. While a PPSI protocol can be obtained by combining existing circuit-PSI and generic circuit-based secure computation, this naive approach is not efficient. Therefore, we also provide a specially designed PPSI protocol with linear complexity and good concrete efficiency. We implemented the protocol and evaluated it with extensive experiments. The results validated the efficacy of our PPSI protocol.

Keywords: Private set intersection · Secure comparison · Secure two-party computation

1 Introduction

Private set intersection (PSI) enables two parties \mathcal{C} and \mathcal{S} who have private input sets \mathbf{x} and \mathbf{y} to get items that they have in common (i.e., $\mathbf{x} \cap \mathbf{y}$) without revealing other information. This technique can be applied to many applications, such as matching in mobile apps [7], threat detection [30] and document search [11]. Due to its versatility, many PSI protocols have been investigated [5–7, 19]. Recent research on PSI focuses on improving its efficiency and the currently most efficient PSI protocols are KKRT [19] and CM20 [5]. Even though PSI can solve many real-world problems, in many other scenarios, a standard PSI protocol is not sufficient because the application's need may not be simply the intersection of two sets. For example:

Advertising Campaigns. A Transaction Data Provider (TP) has a database of the ids of customers and the amount they spend on transactions, denoted as

(*id, spending*) [22]. An Advertisement Company (AdC) only has a database of customer ids of which some have watched its advertisements. Rather than only privately computing the intersection of customer ids to analyze the advertisement conversion rates, AdC also wants to find among those click-through customers, the high-value customers who spent above a threshold. This would allow the AdC to better negotiate the commission and improve its advertisement strategy. In this application, TP’s database needs to be kept private, and AdC’s customers and the threshold are also private information since they relate to the company’s commercial confidentiality.

Database Join. Join is a fundamental operation in databases. With the prevalence of distributed databases and vertical federated learning [24], secure join over distributed data tables becomes notable [20, 26]. Specifically, two database owners each owns a table X and Y , respectively. They want to perform join operations on the primary keys of X and Y to align data in the two tables and filter the results with certain conditions. If we use X_j to denote the j -th column of the table X , an example of an SQL-style join query is given as:

select X_1 from X inner join Y on $X_1 = Y_1$ where $Y_2 > 23.3$

As we can see, in those two applications, we are not interested in outputting the intersection set, but rather a subset of the intersection that meets certain conditions specified as a predicate over the payload associated with the element in the intersection. In the first example, AdC wants to find the customers who have watched the advertisement and also made a purchase on TP, and the amount of purchase is higher than a pre-defined threshold. Similarly, in the second example, X_1 and Y_1 can be seen as two sets, and items in Y_2 are corresponding payloads. The intended output is the set of items in Y , whose primary keys are also in X_1 and payloads satisfy the predicate $Y_2 > 23.3$ (or other arbitrary predicates). Consequently, existing PSI/PSI with payload computation works [3, 5–7, 19, 21, 31, 40] are not enough to meet the requirements of the above-mentioned applications.

Motivated by this, we bring up a new primitive called *Predicate Private Set Intersection* (PPSI). PPSI allows each item on an input set of one party to have an associated payload, and another party can define a predicate over the payload. Then, only those items that are in the intersection of the input sets and whose corresponding payloads meet the constraints will be revealed. A naive way to obtain a PPSI protocol is to use circuit-PSI proposed in [3, 31]. In a circuit-PSI protocol, two parties will receive shares of the intersection set instead of learning the intersection in clear. Then, we can add a predicate computation on the payloads and then feed the computation results and the shares to a circuit, thus achieving the desired functionality of PPSI. However, this naive solution is not efficient. Therefore we need a new design rather than trivially applying circuit-PSI.

Our contribution in this paper can be summarized as follows.

- We present the definition of *Predicate Private Set Intersection* (PPSI) and a concrete construction of the protocol. To avoid costly circuit-based computation, we design a sub-protocol, which we call predicate masking. At a high

level, for two items each from a different input set, the two parties jointly evaluate the predicate over the payload, if the result is true, the same random value will be added to the two set items; if the evaluation result is false, different random values will be added to the two set items. Hence, after that, the two parties only need to perform a plain PSI protocol (e.g., KKRT & CM20) and the intersection computed will contain only the items that meet the condition.

- We demonstrate PPSI with two exemplar applications. Given the applications, we can apply further optimizations. We use a batched secure comparison protocol (BatchComp) as a building block to achieve better efficiency for predicate computation. This sub-protocol may be of independent interest in its own right.
- We evaluate the performance of our PPSI protocol in two applications with extensive experiments. The experimental results demonstrate that PPSI is practical and scalable. We also compare our protocols with the state-of-the-art circuit-PSI protocol CGS [3], and PPSI protocol is $3.9 \sim 10\times$ faster and about $3.1\times$ more communication efficient under different network settings.

The rest of the paper is organized as follows. In Sect. 2, we introduce PSI and its various variants. In Sect. 3, we formally define PPSI and its goals in security and efficiency. Sect. 4 and Sect. 5 present the technical foundation and details of our PPSI protocol, respectively. Sect. 6 theoretically proves the security of PPSI and Sect. 7 assesses its efficiency. Sect. 8 draws the conclusion of this paper.

2 Related Works

PSI was first introduced in [25]. Since then, lots of research about PSI has been carried out [3, 5–8, 10, 15, 19, 31, 33, 37].

PSI. The aforementioned long line of works [5, 10, 19, 29, 33] are based on oblivious transfer to achieve PSI. And those protocols outperform other generic solutions. The first work in this line is [10], which is based on OT extension and a specially designed data structure called garbled Bloom filter. It achieves linear computational/communication complexity, and requires only a few public key operations to bootstrap OT extension. In KKRT [19], the authors constructed a randomized encoding protocol based on OT extension [17]. Based on the same techniques in [19], Pinkas et al. [33] showed a detailed analysis of how different parameters affect the communication cost. Next, Pinkas et al. [29] achieved a half communication cost than that of KKRT, but roughly 6–7 times computation overhead compared to KKRT. Then, in CM20 [5], the authors designed a PSI protocol that is the fastest in a network with moderate bandwidth (e.g., 30–100 Mbps) and outperforms [29] in both computation and communication cost. Nevertheless, KKRT can still be viewed as a protocol optimized for the LAN setting, where bandwidth is not a bottleneck, and it achieves better computation/communication trade-offs. And CM20 targets and achieves better in the middle range bandwidth (e.g., 30–100 Mbps) setting. Some other PSI works [6–8] in this category are based on Homomorphic Encryption

(HE) methods. Chen et al. [7] introduced a PSI protocol based on Fan-Vercauteren leveled fully homomorphic encryption scheme [12] that has a communication complexity logarithmic in the larger input set size. Then, the security model of their work is strengthened in a follow-up paper [6]. Those HE-based protocols focus on computing the intersection of the unbalanced input sets and reducing communication overheads.

Circuit-PSI. The other line of work is circuit-PSI [3, 15, 31, 37]. Huang et al. [15] put forward a notion of circuit-PSI protocol based on garbled circuits, which enables the secure computation of arbitrary functions f over the intersection. Subsequently, some enhanced 2-party protocols [3, 31] for circuit-PSI have been proposed. In circuit-PSI [3, 31], the intersection results are secret-shared between two parties, then these two parties can take the results as inputs of circuits, which achieve the functionality of f . The inputs of those circuits cannot only be the intersection results, but also some associated payloads. Therefore, [3, 31] can compute on the associated payloads of the intersection items without leaking the intersection. And [3, 31] both claim $O(n)$ complexity in the semi-honest setting for PSI with computation. But the circuit-PSI proposed in [3] is state-of-the-art and is 2.3x more communication efficient and around 2.3x faster than the protocol in [31]. We can use the circuit-PSI protocol in [3] to perform the same functionality as PPSI. However, our PPSI protocol achieves much better efficiency than [3]. Besides, in [37], the author proposed a novel protocol, named Conditional Private Set Intersection (CPSI), which can enforce additional conditions for PSI. The authors utilized Trusted Execution Environments (TEEs) and HE to construct their protocol. However, their protocol is subject to the security issues of TEE [36] and the efficiency limitation of HE.

PSI with Payload Computation. Some other works have also considered the application of PSI with payload computation [3, 21, 31, 40]. Those works aim to securely compute some desired functions on the payloads associated with items that are in the intersection of the two input sets, and only reveal the computation results of payloads to parties. For example, PSI-Sum, which has been considered in many works [3, 21, 31, 40], aims to compute the sum of the payloads for the items in the intersection set and only return the sum result to parties. Therefore, in PSI with payload computation, the parties are interested in the payload computation results rather than the items in input sets, which is different from PPSI.

3 Problem Formulation

Before discussing the definition of our focused problem, we first introduce the definition of Predicate Priate Set Intersection (PPSI). Next, we identify the design goal of our protocol.

3.1 Defining PPSI

We first present the definition of PPSI. Suppose one client \mathcal{C} has a set $\mathbf{x} = \{x_1, \dots, x_m\}$ with the constraining value α , and one server \mathcal{S} provides a set $\mathbf{y} =$

$\{y_1, \dots, y_n\}$ with the associated payloads $\mathbf{p}_S = \{p_{S,1}, \dots, p_{S,n}\}$. Rather than merely computing the intersection set $\mathbf{x} \cap \mathbf{y}$, we wish to further confine this result subject to some constraints. Therefore, we define a predicate $\mathbf{P} : (\alpha, p_{S,i}) \rightarrow \{0, 1\}$, for $i \in [1, n]$. This predicate represents the constraint agreed upon by the client \mathcal{C} and the server \mathcal{S} . And it can be replaced by any secure two-party computation protocol. After executing our PPSI protocol, \mathcal{C} gets the results $\mathbf{r} = \{r_1, \dots, r_m\}$. If an item $r_{j, j \in [1, m]} = 1$, there exists an item $y_{i, i \in [1, n]} \in \mathbf{y}$ and $x_j = y_i$, and the associated payloads are subject to the predicate $\mathbf{P}(\alpha, p_{S,i}) = 1$, otherwise $r_j = 0$ and $\mathbf{P}(\cdot, \cdot) = 0$. The ideal functionality $\mathcal{F}_{\text{PPSI}}$ is shown in Fig. 1.

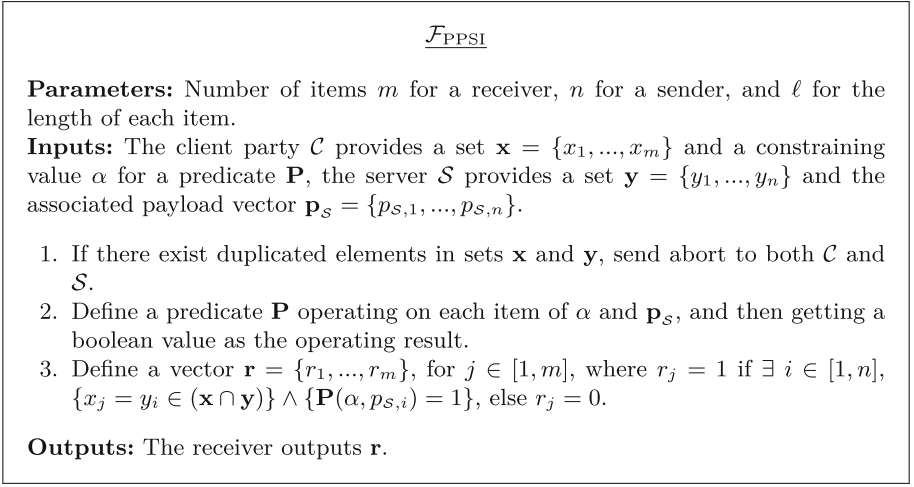


Fig. 1. Ideal functionality $\mathcal{F}_{\text{PPSI}}$.

3.2 Design Goal

In this work, our goal is to design an efficient predicate private set intersection protocol in the semi-honest setting, the following objectives need to be achieved.

- *Privacy preserving:* The inputs of one party in our protocol are kept private from the other party. Meanwhile, the size of the intersection set and access pattern also should be preserved and not revealed to any party.
- *Efficiency:* We also focus on reducing computation and communication costs to make this protocol as efficient as possible, as complex computation inevitably results in higher overhead. For computing set intersection with further computation, it is better to make the computation and communication overhead better than the generic circuit-PSI protocol.

4 Preliminaries

In this section, we will review the cryptographic techniques utilized in this paper, including secret sharing, cuckoo hashing, Oblivious Transfer (OT), secure comparison and the circuit-PSI protocol, and we will give the blueprint of the protocol [3].

4.1 Secret Sharing

Throughout our paper, we use 2-out-of-2 additive secret sharing scheme over the ring \mathbb{Z}_{2^ℓ} . $\langle x \rangle_{\mathcal{C}}$ and $\langle x \rangle_{\mathcal{S}}$ are used to denote the additive shares that belong to party \mathcal{C} and \mathcal{S} respectively. And $\langle x \rangle_{\mathcal{C}}^B$ and $\langle x \rangle_{\mathcal{S}}^B$ represent boolean shares of a binary value x for \mathcal{C} and \mathcal{S} , respectively. Then, we use $\text{Share}(x)$ to denote the algorithm that takes x in \mathbb{Z}_{2^ℓ} as input and outputs the two shares $\langle x \rangle_{\mathcal{C}}$ and $\langle x \rangle_{\mathcal{S}}$ over \mathbb{Z}_{2^ℓ} , and $\langle x \rangle_{\mathcal{C}} + \langle x \rangle_{\mathcal{S}} = x$ (where $+$ denotes addition in \mathbb{Z}_{2^ℓ}). In terms of security, the shares $\langle x \rangle_{\mathcal{C}}$ and $\langle x \rangle_{\mathcal{S}}$ completely hide the secret x . Someone who only has one share cannot infer any information about the secret, since the shares are totally random [9].

4.2 Hashing Techniques

Cuckoo Hashing. Cuckoo hashing [28] is used in our protocol to align all items \mathcal{C} and \mathcal{S} owned and reduce the computation cost. In detail, cuckoo hashing uses some universal hash functions to map n items in a set $\mathbf{x} = \{x_1, \dots, x_n\}$ to a cuckoo hash table \mathbf{T} with b bins. Each bin only contains at most one item. In our protocol, we choose a variant of cuckoo hashing [32], which uses three hash functions h_1, h_2, h_3 . To insert an item x_i into a cuckoo hash table \mathbf{T} , the brief procedure is as follows: 1) Check whether three bins indexed by $h_1(x_i)$, $h_2(x_i)$ and $h_3(x_i)$ have existing items or not; 2) If at least one of those bins is empty, insert x_i into the empty bin with the smallest index; 3) Otherwise, randomly select one bin in $h_1(x_i)$, $h_2(x_i)$ or $h_3(x_i)$, and evict the prior item in the selected bin and place x_i in it. 4) Recursively execute 1) – 3) to insert the evicted item from the previous step into the table \mathbf{T} . This procedure is repeated until all items in \mathbf{x} are inserted and no more evictions are needed. After a certain number of iterations, if there are still some items that cannot be inserted into bins, it will cause failure and abortion to this process. Similar to [3, 33], we set $b = 1.27n$ and achieve a failure probability of less than 2^{-40} . We utilize the formalization in [13]:

$$\mathbf{T}_{\mathbf{x}} \leftarrow \text{Cuckoo}_{h_1, h_2, h_3}^b(\mathbf{x}), \quad (1)$$

where $h_1, h_2, h_3: \{0, 1\}_l \rightarrow [b]$, and all items in \mathbf{x} are inserted into a table $\mathbf{T}_{\mathbf{x}}$ with b bins.

Simple Hashing. In our paper, we use the same hashing functions used in cuckoo hashing (i.e., h_1 , h_2 , and h_3) to achieve simple hashing. To map an item x to a hash table with simple hashing, this item will be put into three bins indexed with $h_1(x)$, $h_2(x)$, and $h_3(x)$. Then, for mapping m items to a hash

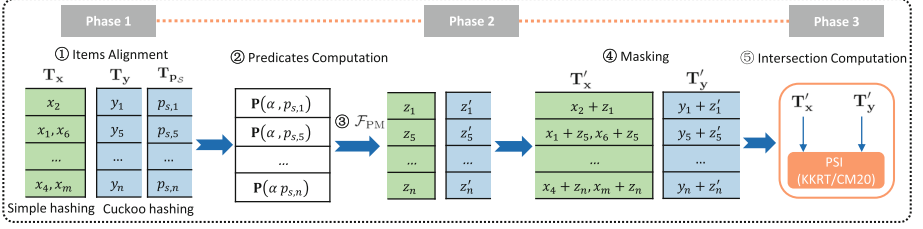


Fig. 2. A toy example of our PPSI protocol.

table with b bins, each item will be put into three bins, and each bin may have multiple items since collisions will happen. We assume the maximum number of items in one certain bin is $maxb$. As suggested in [33], $maxb$ is determined by the size of the input set m , the number of bins b and the statistical parameters via

$$P(\text{"}\exists \text{ bin with } \geq maxb \text{ items"}') \leq b \cdot \left[\sum_{\beta=maxb}^m \binom{m}{\beta} \left(\frac{1}{b}\right)^\beta \left(1 - \frac{1}{b}\right)^{m-\beta} \right], \quad (2)$$

where P represents the possibility and it should be no greater than 2^{-40} .

4.3 Oblivious Transfer

Oblivious Transfer (OT) is an essential primitive in cryptography and can be used for constructing secure computation protocols. It is a two-party protocol between a sender and a receiver. The sender can transfer some of the potentially many pieces of messages to the receiver but remain oblivious as to which pieces have been transferred. We use different types of OT functionalities in this paper. The basic one is 1-out-of-2 OT $\binom{2}{1}$ -OT $_\ell$, where a sender inputs two strings (x_0, x_1) with length ℓ and a receiver inputs a single bit σ . And the receiver receives the string x_σ and learns nothing about $x_{1-\sigma}$ [17]. Then, we use $\binom{n}{1}$ -OT $_\ell$ to denote 1-out-of- n OT, which is extended from the 1-out-of-2 OT. The sender's inputs are n strings (x_0, \dots, x_n) and the receiver receives the exact string x_σ [18]. And $\binom{2}{1}$ -ROT $_\ell$ is the random OT. The sender receives two random elements x_0 and x_1 with length ℓ and the receiver outputs x_σ according to the chosen bit σ [18].

Notably, a large number of OT instances can be efficiently implemented by a few base OT instances with asymmetric key operations and symmetric key operations [17, 18], denoted as IKNP-style OT. As suggested in [16], the general secure two-party computation can be upgraded by using VOLE-style OT extension, so the IKNP-style OT primitives in PPSI can be substituted by VOLE-style OT. We will discuss this in Sect. 7.

4.4 Secure Comparison

A secure comparison protocol takes x and y from two parties as inputs and returns the boolean shares of $\mathbf{1}\{x < y\}$ to each party. Notably, Rathee et al. [34] propose a secure comparison protocol based on a recursive problem-solving approach based on $\binom{n}{1}$ -OT $_\ell$ and the functionality \mathcal{F}_{AND} . \mathcal{F}_{AND} take boolean shares of values $x, y \in \{0, 1\}$ as inputs and returns boolean shares of the comparison result $x < y$. Then the core idea of the secure comparison protocol in CrypTFlow2 [34] is built on an observation

$$\mathbf{1}\{x < y\} = \mathbf{1}\{x^1 < y^1\} \oplus (\mathbf{1}\{x^1 = y^1\} \wedge \mathbf{1}\{x^0 < y^0\}), \quad (3)$$

where $x, y \in \mathbb{Z}_{2^\ell}$, $x = x^1 || x^0$ and $y = y^1 || y^0$. The basic OT protocol used in CrypTFlow2 is IKNP-style OT protocol [18]. For more details and the implementation of this protocol can refer to Appendix 1.

4.5 Technique Overview of Circuit-PSI

Circuit-PSI is a protocol that can be used as a useful implementation of PPSI. To illustrate the difference between our construction and circuit-PSI, we give a general description of the state-of-the-art circuit-PSI protocol [3] in this section.

First, a party \mathcal{C} will use cuckoo hashing to map all items of \mathcal{C} 's input set \mathbf{x} to a hash table $\mathbf{T}_\mathbf{x}$. Then another party \mathcal{S} uses a simple hashing method to hash \mathcal{S} 's set \mathbf{y} to another hash table $\mathbf{T}_\mathbf{y}$ with the same hash functions as used in cuckoo hashing. For security purposes, \mathcal{C} will use dummy items to pad each bin to make sure each bin has the same number of items. Next, \mathcal{C} and \mathcal{S} need to generate random values for each item in the bins of their hash tables. Therefore, if an item is in the intersection of $\mathbf{x} \cap \mathbf{y}$, then this item will exist in the same bin of $\mathbf{T}_\mathbf{x}$ and $\mathbf{T}_\mathbf{y}$. Then, the authors in [3] proposed an Oblivious Programmable Pseudorandom Function (OPPRF) protocol and built a private set membership protocol based on OPPRF. If the table $\mathbf{T}_\mathbf{x}$ and $\mathbf{T}_\mathbf{y}$ both have b bins, for bin $\tau \in [1, b]$, \mathcal{C} and \mathcal{S} perform a private set membership protocol over the bins with the same indexes. \mathcal{C} and \mathcal{S} compare all items in $\mathbf{T}_\mathbf{x}[\tau]$ and $\mathbf{T}_\mathbf{y}[\tau]$ to get whether the item in $\mathbf{T}_\mathbf{x}[\tau]$ exists in $\mathbf{T}_\mathbf{y}[\tau]$. After this protocol, \mathcal{C} and \mathcal{S} will get the secret-shared boolean results, which can be used as input for other subsequent computations (e.g., circuit-based computation, associated payloads computation).

5 The Construction of PPSI

In this section, we present our PPSI protocol. Before delving into the details, we first give a high-level overview of PPSI, which achieves the ideal functionality of PPSI. Then we introduce the technical description of PPSI and the details of our main sub-protocols. Finally, we discuss the applications of PPSI.

5.1 High-Level Overview

In our PPSI protocol, first, the party \mathcal{S} will use cuckoo hashing to map all items of \mathcal{S} 's input set \mathbf{y} and payload set to a hash table. Then the other party \mathcal{C} uses a simple hashing method to hash all items in \mathbf{x} to another hash table with the same hash functions as used in cuckoo hashing. The process can align all items of \mathbf{x} and \mathbf{y} . Then, \mathcal{C} and \mathcal{S} perform a secure two-party computation according to the predicate \mathbf{P} . The inputs of \mathbf{P} are the constraining value α from \mathcal{C} and payloads from \mathcal{S} . The outputs of \mathbf{P} are boolean values and secret-shared to \mathcal{C} and \mathcal{S} . If the boolean value is equal to 1, \mathcal{C} gets a random value z and \mathcal{S} will get z' , where $z = z'$. If the boolean value is equal to 0, \mathcal{C} and \mathcal{S} also get z and z' respectively, while $z \neq z'$. Next, \mathcal{C} and \mathcal{S} can add those random values z and z' to the items in their sets. Therefore, if an item s is in the intersection set $\mathbf{x} \cap \mathbf{y}$, and the corresponding payload of the item is p and $\mathbf{P}(\alpha, p) = 1$, then \mathcal{C} and \mathcal{S} will add z and z' to this item in their sets to get new sets. It is obvious that $s + z = s + z'$ if $z = z'$. Finally, \mathcal{C} and \mathcal{S} will perform a plain PSI protocol on their new sets. Therefore, this process can be seen as a filter in PPSI, and it can filter items that are in the intersection but do not satisfy the predicate.

5.2 Technical Description

In this section, we present a description of our PPSI protocol. We assume two parties (\mathcal{C} and \mathcal{S}) need to perform PPSI, the process of PPSI can be divided into three phases as follows.

Phase 1) Items Alignment. In this phase, two parties use hashing techniques to align their input items. Similar to circuit-PSI protocols, two parties first map their items to hash tables, which consist of multiple bins. If an input item is in the intersection result, both parties map it to the same bin. This process can reduce the computation cost of subsequent phases. To be specific, \mathcal{S} uses cuckoo hashing to hash the items in \mathbf{y} into a hash table \mathbf{T}_y , i.e., $\mathbf{T}_y \leftarrow \text{Cuckoo}_{h_1, h_2, h_3}^b(\mathbf{y})$. And the table \mathbf{T}_y has b bins. According to the definition of cuckoo hashing, there is only one item in each bin of \mathbf{T}_y . Then, \mathcal{C} uses simple hashing functions h_1 , h_2 and h_3 to hash the items in \mathbf{x} into a two-dimension hash table \mathbf{T}_x with b bins. That is, an item $x_j \in \mathbf{x}$ will exist in three hash bins indexed by $h_1(x_j)$, $h_2(x_j)$ and $h_3(x_j)$. In each bin of \mathbf{T}_x , there will be multiple items since collisions will happen. In terms of privacy concerns, if any two or three of $h_1(x_j)$, $h_2(x_j)$ and $h_3(x_j)$ collide and are mapped to one bin, then \mathcal{C} needs to map one or two dummy items any other bin to ensure that \mathcal{C} maps $3m$ items in all bins. Otherwise, \mathcal{S} would know that a collision has happened when \mathcal{C} aligns his/her items. We assume the bin $\mathbf{T}_x[\tau]$ has $maxb_\tau$ items, where $\tau \in [1, b]$. Next, \mathcal{S} also maps the associated payloads to a hash table \mathbf{T}_{p_S} with the same mapping method. That is, if items $y_j \in \mathbf{y}$ appear in the bin $h_2(y_j)$, then the associated payload $p_{S,j}$ is also in the same bin $h_2(y_j)$ of \mathbf{T}_{p_S} . At the end of this phase, \mathcal{C} and \mathcal{S} have aligned all items they own. For an item in the intersection $\mathbf{x} \cap \mathbf{y}$, this item will be mapped into the same bins of \mathbf{T}_x and \mathbf{T}_y . Therefore, \mathcal{C} and \mathcal{S} only need to compare the items in bins with the same indexes in \mathbf{T}_x and \mathbf{T}_y .

Phase 2) Predicate Masking. In this phase, a predicate \mathbf{P} operating on the constraining value α and payloads \mathbf{p}_S is computed between \mathcal{C} and \mathcal{S} . The predicate \mathbf{P} takes α from \mathcal{C} and a payload from $\mathbf{T}_{\mathbf{p}_S}$ as inputs, and outputs boolean results for this payload, indicating whether this payload meets the constraints or not. We assume $\tau \in [1, b]$, and compute $\mathbf{P}(\alpha, \mathbf{T}_{\mathbf{p}_S}[\tau]) \rightarrow p_\tau^*$ and $p_\tau^* \in \{0, 1\}$. And \mathbf{P} could be any secure two-party computation protocol executed between \mathcal{C} and \mathcal{S} , as long as the final results are boolean values and secret-shared to \mathcal{C} and \mathcal{S} . We will give three applications of different secure two-party protocols in Sect. 5.5.

After \mathcal{C} and \mathcal{S} have executed the predicate \mathbf{P} , it will generate a boolean value for the payload in each bin of $\mathbf{T}_{\mathbf{p}_S}$. And this boolean value is secret-shared between \mathcal{C} and \mathcal{S} . Then, \mathcal{C} and \mathcal{S} need to call a predicate masking functionality \mathcal{F}_{PM} that we designed to securely compute a random value for masking the items in each bin of \mathbf{T}_x and \mathbf{T}_y according to the result of \mathbf{P} . The functionality of \mathcal{F}_{PM} is formalized as Fig. 3. Our \mathcal{F}_{PM} takes as input boolean shares of choice bits p_τ^* , and returns two random values z_τ and z'_τ to \mathcal{C} and \mathcal{S} . If the corresponding choice bit is equal to 1, then $z_\tau = z'_\tau$, else $z_\tau \neq z'_\tau \stackrel{\$}{\leftarrow} \mathbb{Z}_{2^\ell}$. This random value leaks no information about the inputs. Next, for $\tau \in [1, b]$ and $\beta \in [1, \max b_\tau]$, \mathcal{C} adds z_τ to the item $\mathbf{T}_x[\tau][\beta]$, and maps this new item to the bin $\mathbf{T}'_x[\tau]$ of a new two-dimension hash table \mathbf{T}'_x . At the same time, \mathcal{S} adds z'_τ to the item $\mathbf{T}_y[\tau]$, and maps those new items into the same bin of a new hash table \mathbf{T}'_y .

The core idea of this phase is that, if an item $x_j \in \mathbf{x}$ is equal to the item $y_i \in \mathbf{y}$ and the associated payload meets the constraints $p_i^* = 1$, then we add the same random value to both x_j and y_i . Otherwise, we add different random values to x_j and y_i . After that, we can filter out the items that are in the intersection but whose payloads do not meet the constraint predicate \mathbf{P} .

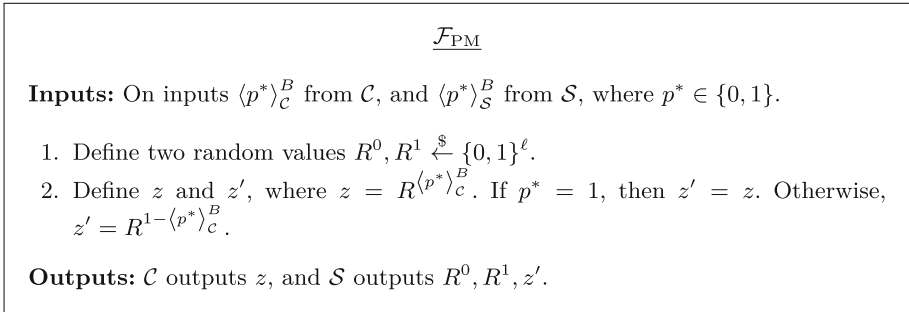


Fig. 3. Ideal functionality \mathcal{F}_{PM} .

Phase 3) Intersection Computation. In this phase, \mathcal{C} performs a plain PSI protocol (KKRT [19] or CM20 [5]) protocol with \mathcal{S} . \mathcal{C} and \mathcal{S} take the items in $\mathbf{T}'_{\mathbf{x}}$ and $\mathbf{T}'_{\mathbf{y}}$ as inputs to PSI. And there are n items in $\mathbf{T}'_{\mathbf{y}}$ and $3m$ items in $\mathbf{T}'_{\mathbf{x}}$. It is worth noting that, we consider the scenario of \mathcal{C} being a receiver. However, our PPSI protocol is flexible. \mathcal{S} can also be set as a receiver, who gets the final result. And \mathcal{C} plays the role of a sender. Therefore, this flexibility enables our protocol to meet more practical application requirements.

A Toy Example. We give an example of PPSI in Fig. 2. All green tables represent the hash tables of \mathcal{C} , and blue tables belong to \mathcal{S} . First, they map all the items and payloads in $\mathbf{T}_{\mathbf{x}}$, $\mathbf{T}_{\mathbf{y}}$, $\mathbf{T}_{\mathbf{p}_{\mathcal{C}}}$ and $\mathbf{T}_{\mathbf{p}_{\mathcal{S}}}$. Second, \mathcal{C} and \mathcal{S} perform a predicate computation for each payload and the constraining value α . Third, they invoke the PM protocol to get pairs of random values. If $\mathbf{P}(\alpha, p_{\mathcal{S},1}) = 1$, then $z_1 = z'_1$. Next, \mathcal{C} and \mathcal{S} mask their items with those random values, and input the items in $\mathbf{T}_{\mathbf{x}}$ and $\mathbf{T}_{\mathbf{y}}$ to a PSI protocol. Consequently, if $x_2 = y_1$ and $\mathbf{P}(\alpha, p_{\mathcal{S},1}) = 1$, we can get $x_2 + z_1 = y_1 + z'_1$ and this value is still in the intersection set.

$\underline{\mathcal{F}_{\text{BatchComp}}}$

Parameters: Size of a vector n .

Inputs: On inputs $\alpha \geq 0$ from \mathcal{C} , and $\mathbf{p} = \{p_1, \dots, p_n\}$ and from \mathcal{S} .

1. Define a vector $\mathbf{p}^* = \{p_1^*, \dots, p_n^*\}$, where $p_i^* = 1$ if $p_i > \alpha$ and $p_i^* = 0$ otherwise.
2. Get the boolean shared vectors $\langle \mathbf{p}^* \rangle_{\mathcal{C}}^B$ and $\langle \mathbf{p}^* \rangle_{\mathcal{S}}^B$ from $\text{Share}(\mathbf{p}^*)$.

Outputs: \mathcal{C} outputs $\langle \mathbf{p}^* \rangle_{\mathcal{C}}^B$, and \mathcal{S} outputs $\langle \mathbf{p}^* \rangle_{\mathcal{S}}^B$.

Fig. 4. Ideal functionality $\mathcal{F}_{\text{BatchComp}}$.

5.3 The Predicate Masking Protocol

In this section, we will introduce our Predicate Masking protocol (PM). It achieves the ideal functionality \mathcal{F}_{PM} as shown in Fig. 3. This protocol aims to generate random values for two parties according to a secret-shared boolean value. Then, two parties can use random values output by \mathcal{F}_{PM} to mask their own items for other computation. Therefore, this protocol can be used as a building block in other secure two-party computation.

Next, we give a detailed description of our protocol for realizing \mathcal{F}_{PM} . When \mathcal{C} has a boolean value $\langle p^* \rangle_{\mathcal{C}}$, and \mathcal{S} gets $\langle p^* \rangle_{\mathcal{S}}$, they invoke the functionality \mathcal{F}_{PM} as shown in Algorithm 1. As we can see, the main part of this algorithm is based on a ROT protocol. At the end of this algorithm, \mathcal{C} outputs z , and \mathcal{S}

Algorithm 1. Predicate Masking, \mathcal{F}_{PM}

Input: \mathcal{C} holds a boolean value $\langle p^* \rangle_{\mathcal{C}}^B$;
 \mathcal{S} holds a boolean vector $\langle p^* \rangle_{\mathcal{S}}^B$.

Output: \mathcal{C} learns z , and \mathcal{S} learns z' , s.t., $z = z'$ if $p^* = 1$, else $z_r, z'_r \xleftarrow{\$} \{0, 1\}^\ell$.

- 1: \mathcal{C} and \mathcal{S} invoke an instance of $\binom{2}{1}$ -ROT where \mathcal{C} is the receiver with a choose bit $\langle p^* \rangle_{\mathcal{C}}^B$ and \mathcal{S} is the sender. Then, \mathcal{S} gets $R^0, R^1 \xleftarrow{\$} \mathbb{Z}_2^\ell$, and \mathcal{C} receives $R^{\langle p^* \rangle_{\mathcal{C}}^B}$.
- 2: \mathcal{C} sets $z = R^{\langle p^* \rangle_{\mathcal{C}}^B}$.
- 3: **if** $\langle p^* \rangle_{\mathcal{S}}^B = 0$ **then**
- 4: \mathcal{S} sets $z' = R^1$.
- 5: **else**
- 6: \mathcal{S} sets $z' = R^0$.
- 7: **end if**
- 8: \mathcal{C} outputs z ; \mathcal{S} outputs z' .

gets z' . If \mathcal{C} and \mathcal{S} want to call this functionality n times to generate n pairs of different random values, it can be efficiently achieved by one round of end-to-end communication. The security of this protocol is based on the ROT protocol.

5.4 The Batched Secure Comparison Protocol

The Batched Secure Comparison protocol (BatchComp) is to realize batch secure comparisons at one time. Consider a case where \mathcal{C} inputs a variable α and \mathcal{S} inputs a set $\{p_1, \dots, p_n\}$. And they wish to obtain the boolean shared values of the comparison results $\{\mathbf{1}\{\alpha < p_1\}, \dots, \mathbf{1}\{\alpha < p_n\}\}$. We present the ideal functionality of BatchComp protocol in Fig. 4. Inspired from [17, 27], instead of repeating n times secure comparison protocol in CryptFlow2 [34], we propose an efficient secure comparison protocol (BatchComp) to do those secure comparisons at once.

As defined in Fig. 4, after this protocol, \mathcal{C} and \mathcal{S} will get a boolean share of a vector, and each item of this vector indicates whether each item of \mathcal{S} 's set is bigger than α . Here, we give a detailed explanation of our BatchComp protocol as Algorithm 5.3. There are four stages for our BatchComp protocol:

Splitting Stage: In step 1–2, \mathcal{C} and \mathcal{S} splitting their input values as q parts. And we assume $q = \ell/k$. Therefore, there are 2^k possibilities for each part of input values;

Masking Stage: In steps 3–8, \mathcal{S} chooses two random values for all the parts of its input sets \mathbf{p} , and masks the random values with a bit value;

Choosing Stage: In steps 9–13, \mathcal{C} and \mathcal{S} invoke the $\binom{n}{1}$ -OT $_\ell$ extension protocol, and \mathcal{C} chooses corresponding random values according to his/her input. This stage can compute the shares of the inequalities $\langle lt_{0,j}^t \rangle_{\mathcal{C}, \mathcal{S}}$ and equalities $\langle eq_{0,j}^t \rangle_{\mathcal{C}, \mathcal{S}}$ of the values at the leaf level;

Merging Stage: Based on the same idea in Eq. 3 above, recursively compute the shares of the inequalities and equalities of each node (steps 14–19). Then the

Algorithm 2. Batched Comparison, $\mathcal{F}_{\text{BatchComp}}$

Input: \mathcal{C} and \mathcal{S} hold variable $\alpha \in \{0, 1\}^\ell$ and variables $\mathbf{p} = \{p_1, \dots, p_n\} \in \{0, 1\}^\ell$, respectively.

Output: \mathcal{C} and \mathcal{S} learn $\{\langle 1\{\alpha < p_1\}\rangle_{\mathcal{C}}, \dots, \langle 1\{\alpha < p_n\}\rangle_{\mathcal{C}}\}$ and $\{\langle 1\{\alpha < p_1\}\rangle_{\mathcal{S}}, \dots, \langle 1\{\alpha < p_n\}\rangle_{\mathcal{S}}\}$, respectively.

- 1: \mathcal{C} parses its input as $\alpha = \alpha^{q-1} || \dots || \alpha^0$, and for $j \in [1, n]$, \mathcal{S} parses its inputs as $p_j = p_j^{q-1} || \dots || p_j^0$, where $\alpha^t, p_j^t \in \{0, 1\}^k$, for $t \in [1, q-1]$.
 - 2: $q = \ell/k$, $K = 2^k$.
 - 3: **for** $t = \{0, \dots, q-1\}$ **do**
 - 4: **for** $j = \{1, \dots, n\}$ **do**
 - 5: \mathcal{S} samples $\langle \text{lt}_{0,j}^t \rangle_{\mathcal{S}}, \langle \text{eq}_{0,j}^t \rangle_{\mathcal{S}} \xleftarrow{\$} \{0, 1\}$.
 - 6: **for** $u = \{0, \dots, K-1\}$ **do**
 - 7: \mathcal{S} sets $s_{j,u}^t = \langle \text{lt}_{0,j}^t \rangle_{\mathcal{S}} \oplus 1\{u < y_j^t\}$.
 - 8: \mathcal{S} sets $v_{j,u}^t = \langle \text{eq}_{0,j}^t \rangle_{\mathcal{S}} \oplus 1\{u = y_j^t\}$.
 - 9: **end for**
 - 10: **end for**
 - 11: $\mathcal{C} \& \mathcal{S}$ invoke an instance of $\binom{K}{1}$ -OT where \mathcal{S} is the sender with inputs $\{s_{1,u}^t || \dots || s_{n,u}^t\}_{u \in [0, K-1]}$, and \mathcal{C} is the receiver with input α^t . Then, \mathcal{C} receives $\{s_{1,\alpha^t}^t || \dots || s_{n,\alpha^t}^t\}$.
 - 12: $\mathcal{C} \& \mathcal{S}$ invoke an instance of $\binom{K}{1}$ -OT where \mathcal{S} is the sender with inputs $\{v_{1,u}^t || \dots || v_{n,u}^t\}_{u \in [0, K-1]}$ and \mathcal{C} is the receiver with input α^t . Then, \mathcal{C} receives $\{v_{1,\alpha^t}^t || \dots || v_{n,\alpha^t}^t\}$.
 - 13: **for** $j = \{1, \dots, n\}$ **do**
 - 14: \mathcal{C} sets $\langle \text{lt}_{0,j}^t \rangle_{\mathcal{C}} \leftarrow s_{j,\alpha^t}^t$.
 - 15: \mathcal{C} sets $\langle \text{eq}_{0,j}^t \rangle_{\mathcal{C}} \leftarrow v_{j,\alpha^t}^t$.
 - 16: **end for**
 - 17: **end for**
 - 18: **for** $j = \{1, \dots, n\}$ **do**
 - 19: **for** $i = \{1, \dots, \log q\}$ **do**
 - 20: **for** $t = \{0, \dots, (q/2^i) - 1\}$ **do**
 - 21: \mathcal{C} invokes \mathcal{F}_{AND} with inputs $\langle \text{lt}_{i-1,j}^{2t} \rangle_{\mathcal{C}}$ and $\langle \text{eq}_{i-1,j}^{2t+1} \rangle_{\mathcal{C}}$ to learn output $\langle \text{temp} \rangle_{\mathcal{C}}$.
 - 22: \mathcal{S} invokes \mathcal{F}_{AND} with inputs $\langle \text{lt}_{i-1,j}^{2t} \rangle_{\mathcal{S}}$ and $\langle \text{eq}_{i-1,j}^{2t+1} \rangle_{\mathcal{S}}$ to learn output $\langle \text{temp} \rangle_{\mathcal{S}}$.
 - 23: \mathcal{C} sets $\langle \text{lt}_{i,j}^t \rangle_{\mathcal{C}} = \langle \text{lt}_{i-1,j}^{2t+1} \rangle_{\mathcal{C}} \oplus \langle \text{temp} \rangle_{\mathcal{C}}$.
 - 24: \mathcal{S} sets $\langle \text{lt}_{i,j}^t \rangle_{\mathcal{S}} = \langle \text{lt}_{i-1,j}^{2t+1} \rangle_{\mathcal{S}} \oplus \langle \text{temp} \rangle_{\mathcal{S}}$.
 - 25: \mathcal{C} invokes \mathcal{F}_{AND} with inputs $\langle \text{eq}_{i-1,j}^{2t} \rangle_{\mathcal{C}}$ and $\langle \text{eq}_{i-1,j}^{2t+1} \rangle_{\mathcal{C}}$ to learn output $\langle \text{eq}_{i,j}^t \rangle_{\mathcal{C}}$.
 - 26: \mathcal{S} invokes \mathcal{F}_{AND} with inputs $\langle \text{eq}_{i-1,j}^{2t} \rangle_{\mathcal{S}}$ and $\langle \text{eq}_{i-1,j}^{2t+1} \rangle_{\mathcal{S}}$ to learn output $\langle \text{eq}_{i,j}^t \rangle_{\mathcal{S}}$.
 - 27: **end for**
 - 28: **end for**
 - 29: \mathcal{C} sets $\langle p_j^* \rangle_{\mathcal{C}} = \langle 1\{\alpha < p_j\}\rangle_{\mathcal{C}} \leftarrow \langle \text{lt}_{\log q,j}^0 \rangle_{\mathcal{C}}$, and \mathcal{S} sets $\langle p_j^* \rangle_{\mathcal{S}} = \langle 1\{\alpha < p_j\}\rangle_{\mathcal{S}} \leftarrow \langle \text{lt}_{\log q,j}^0 \rangle_{\mathcal{S}}$
 - 30: **end for**
-

values of the roots indicate the final output (step 20). Therefore, \mathcal{C} and \mathcal{S} get the shared values of the comparison results $\mathbf{p}^* = \{1\{\alpha < p_1\}, \dots, 1\{\alpha < p_n\}\}$.

The core idea behind our protocol is that we replace the $\binom{K}{1}$ -OT $_\ell$ protocol in the **Choosing stage** with a batched OT protocol proposed in [27]. As we can see from this algorithm, in step 9, \mathcal{C} has a choose bit α^t and \mathcal{S} has a $n \times K$ matrix $\{s_{1,u}^t || \dots || s_{n,u}^t\}_{u \in [1, K-1]}$, then \mathcal{C} and \mathcal{S} need to perform the $\binom{K}{1}$ -OT $_\ell$ protocol n times if we use the secure comparison protocol in CrypTFlow2 [34]. However, for the fixed choices, we only need to perform the $\binom{K}{1}$ -OT $_\ell$ protocol once if we use the batch OT protocol [27]. Therefore, we can reduce half of the bandwidth requirement and make a 2.1x running time improvement than [34].

5.5 Applications of PPSI

In this section, we will give two general applications of our PPSI protocol as mentioned in Sect. 1. The first application is when the predicate \mathbf{P} is a comparison computation, and the second is when \mathbf{P} is a combination of multiple constraints.

Application 1: Advertising Campaigns. An AdC (\mathcal{C}), who has a database of customers id, wants to launch a query on a transaction database of a TP (\mathcal{S}). AdC wants to find those customers who have seen the advertisement and also purchased on TP and the purchased amount is greater than α . We modelize this problem as: \mathcal{C} has a query set $\mathbf{x} = \{x_1, \dots, x_m\}$, a payload value α and \mathcal{S} provides a set $\mathbf{y} = \{y_1, \dots, y_n\}$, and for $i \in [1, n]$, each item y_i has an associate payload $\mathbf{p}_s = \{p_{s,1}, \dots, p_{s,n}\}$, and the predicate \mathbf{P} is defined as a secure comparison computation. After executing our PPSI protocol, \mathcal{C} gets the query result $\mathbf{r} = \{r_1, \dots, r_m\}$, where each item r_j indicates whether x_j is in the intersection set $\mathbf{x} \cap \mathbf{y}$ and the associate payload of it is bigger than the constraining value α . For $j \in [1, m]$ and $i \in [1, n]$, then we enumerate all possible cases:

$$\begin{cases} \text{if } \exists y_i : (x_j = y_i) \wedge (\alpha < p_{s,i}), \text{ then } r_j = 1; \\ \text{if } \exists y_i : (x_j = y_i) \wedge (\alpha \geq p_{s,i}), \text{ then } r_j = 0; \\ \text{if } \forall y_i : (x_j \neq y_i) \wedge (\alpha < p_{s,i}), \text{ then } r_j = 0; \\ \text{if } \forall y_i : (x_j \neq y_i) \wedge (\alpha \geq p_{s,i}), \text{ then } r_j = 0. \end{cases}$$

An Optimization. We propose an optimization method when \mathcal{C} and \mathcal{S} perform the PPSI protocol in this application. After \mathcal{C} and \mathcal{S} align all items and payloads, they need to perform a secure comparison protocol with their inputs α and $p_{s,i}$. We can use the state-of-the-art secure comparison protocol proposed in CrypTFlow2 [34]. Therefore, \mathcal{C} and \mathcal{S} invokes the secure comparison protocol of CrypTFlow2 b times. As we can see, α stays unchanged in those b times comparisons. Then, instead of running b times secure comparison protocol, we propose a Batched Secure Comparison protocol (BatchComp), which can do b times secure comparison in one time. The ideal functionality of BatchComp $\mathcal{F}_{\text{BatchComp}}$ is shown as Fig. 4. And we give the details of $\mathcal{F}_{\text{BatchComp}}$ in Sect. 5.4.

Application 2: Database Join. In this application, we give an example of database join. To demonstrate different situations, this example is a combination of multiple predicates. For example, an SQL-styled query is as follows:

select X_1 **from** X **inner join** Y **on** $X_1 = Y_1$ **where** $\alpha < Y_2 < \gamma$

In this example, the predicates are defined as determining whether the payloads are in a certain range $[\alpha, \gamma]$. We can divide this constraining range into two separate predicates: the first is to compare α and Y_2 , and the second is to compare γ and Y_2 . Therefore, we can perform the **Phase 2** of PPSI two times to add different pairs of random values to the items of input sets. We formalize this problem as follows:

Suppose \mathcal{C} has a query set $\mathbf{x} = \{x_1, \dots, x_m\}$, two values α and γ , which defined a constraining range $[\alpha, \gamma]$, \mathcal{S} provides a set $\mathbf{y} = \{y_1, \dots, y_n\}$ and an associated payload $\mathbf{p}_s = \{p_{s,1}, \dots, p_{s,n}\}$. \mathcal{C} aims to get the result $\mathbf{r} = \{r_1, \dots, r_m\}$, where each item r_j indicates whether x_j is in the intersection set $\mathbf{x} \cap \mathbf{y}$ and the associate payload of it is between the constraining range $[\alpha, \gamma]$. Then \mathcal{C} and \mathcal{S} run PPSI. The same as depicted in Sect. 5, in **Phase 1**, \mathcal{C} and \mathcal{S} will align all items with simple hashing and cuckoo hashing to get \mathbf{T}_x , \mathbf{T}_y and \mathbf{T}_{p_s} (b bins), respectively. The only difference is in **Phase 2**, \mathcal{C} and \mathcal{S} will take secure comparison protocol as predicate \mathbf{P} and perform BatchComp to securely compare the value α and the item in \mathbf{p}_s to get the secret-shared values of boolean results $\mathbf{p}_1^* = \{p_{1,\tau}^*\}_{\tau=1}^b$, where $p_{1,\tau}^* = \mathbf{1}\{\alpha < \mathbf{T}_{p_s}[\tau]\}$. Next, \mathcal{C} and \mathcal{S} invoke the PM protocol to get b random values $z_{1,\tau}$ and $z'_{1,\tau}$ ($\tau \in [1, b]$) respectively, and add those random values to their corresponding items in $\mathbf{T}_x[\tau]$ and $\mathbf{T}_y[\tau]$. Then, \mathcal{C} and \mathcal{S} will perform BatchComp again with different inputs to check whether all the payload values are less than γ . That is, \mathcal{C} and \mathcal{S} securely compute $p_{2,\tau}^* = \mathbf{1}\{\gamma > \mathbf{T}_{p_s}[\tau]\}$ to get $\mathbf{p}_2^* = \{p_{2,\tau}^*\}_{\tau=1}^b$, and then perform PM protocol with the input \mathbf{p}^* . After \mathcal{C} gets b random values $z_{2,\tau}$ and \mathcal{S} gets $z'_{2,\tau}$ ($\tau \in [1, b]$), \mathcal{C} and \mathcal{S} add those random values to the corresponding items in $\mathbf{T}_x[\tau]$ and $\mathbf{T}_y[\tau]$ again. At last, \mathcal{C} and \mathcal{S} perform PSI with inputs \mathbf{T}_x and \mathbf{T}_y in **Phase 3**.

As we can see from this application, the different predicates are constraints for those items in the intersection set. Therefore, we only need to perform all protocols in **Phase 2** multiple times to filter those items whose payloads cannot meet the constraints.

6 Security Analysis

In this section, we analyze the security of the proposed sub-protocols. Under the assumption that all parties are semi-honest, we use simulation-based to prove the security of PPSI. Next, we show that our protocols satisfy our design goals. Under the honest-but-curious assumption, the adversary is allowed to corrupt the client \mathcal{C} or the server \mathcal{S} . To prove that a protocol is secure, the view of the corrupted party is simulatable by given its input and output of this protocol [14, 23].

Definition 1. A protocol is secure and can achieve the functionality \mathcal{F} if there exists a probabilistic polynomial-time simulator Sim that can generate a view for the adversary Adv in the real world and the view is computationally indistinguishable from its view in the ideal world.

To prove the security of our protocols, we then give the important lemma used in our security analysis.

Lemma 1. For a random element $r \xleftarrow{\$} \mathbb{Z}_{2^\ell}$ and any independent element $r' \in \mathbb{Z}_{2^\ell}$, $r \pm r'$ is uniformly random and independent from the element r' .

The complete proof of this lemma can refer to [2, 14]. Based on Definition 1 and Lemma 1, we prove that our sub-protocols can be perfectly simulated as follows:

Theorem 1. The Predicate Masking protocol is secure against semi-honest adversaries.

The proof of this theorem is presented in Appendix 2.

Theorem 2. The Batched Secure Comparison protocol is secure against semi-honest adversaries.

The proof of this theorem is shown in Appendix 2.

Privacy Preserving: As we can deduce from the security proof, the inputs of \mathcal{C} and \mathcal{S} are kept private from each other in our protocol. Meanwhile, the size of intersection set $\mathbf{x} \cap \mathbf{y}$ are kept private from \mathcal{C} and \mathcal{S} , since all intermediate results are secure-shared between \mathcal{C} and \mathcal{S} . Besides, when \mathcal{C} launches a query in the database of \mathcal{S} , all access pattern are kept private from \mathcal{S} since each item in \mathcal{S} are involved in the computation and \mathcal{S} can not know which items that \mathcal{C} gets.

Table 1. The running time in s and communication in MB of the protocol PM (IKNP-style OT based).

n		2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
Time	LAN	0.003	0.012	0.026	0.092	0.29
	WAN	0.32	0.44	0.82	2.34	8.5
Comm.		0.5	1.5	5.25	20.5	81.5

Table 2. The running time in s and communication in MB of the protocol PM (VOLE-style OT based).

n		2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
Time	LAN	0.001	0.004	0.001	0.044	0.157
	WAN	0.081	0.082	0.088	0.113	0.255
Comm.		0.002	0.009	0.04	0.16	0.64

7 Performance Evaluation

In this section, we first give the details about the environment of our experiments and the parameters used in our protocol. Then, we evaluate the building blocks of PPSI. Final, we evaluate the efficiency of PPSI and compare it with the state-of-art circuit-PSI protocol and other PSI-related protocols from different aspects.

Table 3. The running time in s and communication in MB of BatchComp and the secure comparison protocol of CryptFlow2 in the LAN setting, and the items for comparison are 32-bit long. IKNP represents the IKNP-style OT based BatchComp protocol, and VOLE represents the VOLE-style OT based BatchComp protocol.

		n	2^{10}	2^{15}	2^{20}
Time	CryptFlow2		0.014	0.21	6.81
	BatchComp (IKNP)		0.006	0.1	3.32
	BatchComp (VOLE)		0.004	0.058	2.24
Comm.	CryptFlow2		0.47	15.05	481.5
	BatchComp (IKNP)		0.23	7.05	225.5
	BatchComp (VOLE)		0.04	1.12	36.57

Table 4. The running time in s of the first application of PPSI, and the BatchComp and PM protocols are based on VOLE-style OT.

m vs n		2^{12} vs 2^{14}	2^{13} vs 2^{14}	2^{14} vs 2^{14}	2^{12} vs 2^{18}	2^{13} vs 2^{18}	2^{18} vs 2^{18}	2^{12} vs 2^{22}	2^{13} vs 2^{22}	2^{22} vs 2^{22}
Breakdown										
LAN	Alignment	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	0.023	0.024	0.024	1.18	1.18	1.22
	$\mathbf{P} + \mathcal{F}_{PM}$	0.007			0.087			1.39		
	PSI(KKRT)	0.037	0.046	0.062	0.2	0.2	0.69	2.98	3.03	14.1
	PSI(CM20)	0.23	0.31	0.33	0.98	0.97	4.52	13.6	97.7	54.94
WAN	Alignment	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	0.023	0.024	0.024	1.18	1.18	1.22
	$\mathbf{P} + \mathcal{F}_{PM}$	0.4			0.77			3.91		
	PSI(KKRT)	1.12	1.27	1.41	1.87	2.1	8.1	12.72	12.86	113.63
	PSI(CM20)	1.17	1.3	1.58	2.14	2.09	6.39	14.95	17.03	102.27

Table 5. The running time in s and communication in MB of the first application of PPSI, and the BatchComp protocol and PM protocol are based on IKNP-style OT.

m vs n		2^{12} vs 2^{14}	2^{13} vs 2^{14}	2^{14} vs 2^{14}	2^{12} vs 2^{18}	2^{13} vs 2^{18}	2^{18} vs 2^{18}	2^{12} vs 2^{22}	2^{13} vs 2^{22}	2^{22} vs 2^{22}	
Time	LAN	KKRT	0.052	0.06	0.077	0.38	0.38	0.86	6.78	6.83	17.91
		CM20	0.24	0.32	0.35	1.16	1.15	4.7	17.35	17.44	110.16
	WAN	KKRT	2.1	2.25	2.71	4.23	4.47	10.46	35.54	27.18	136.49
		CM20	2.15	2.28	2.89	4.5	4.44	8.75	27.77	31.36	125.13
Comm.	KKRT	2.69	3.97	5.21	20.88	22.91	81.31	317.87	319.15	1303.99	
	CM20	2.31	3.5	4.65	16.31	17.78	73.01	237.87	239.1	1181.13	

7.1 Implementation Details

Our PPSI protocol is implemented in C++ and the code is available at <https://www.ppsi.cn>. For the cuckoo hashing method used in our protocol, we adopt the parameter used in [3, 31]. When mapping n items to a hash table with b bins via three hash functions, we set $b = 1.27n$ for the stash-less setting. In addition, we test two different OT protocols to implement our protocols. One is IKNP-style OT protocol [18], and another is VOLE-style OT protocol [39], which achieves better performance in some circumstances. The statistical security parameter in our implementation is $\sigma = 40$, and the computational security parameter is $\lambda = 128$.

Table 6. Running time in seconds of PPSI and CGS (CGS-PSM1, CGS-PSM2) [3]. We set the sizes of sets are equal and items in sets are of 64-bit length.

Network Setting	LAN					WAN				
	$m = n$	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{14}	2^{16}	2^{18}	2^{20}
CGS-PSM1	0.7	1.65	6.07	24.78	100.12	7.73	16.49	42.85	162.61	652.35
CGS-PSM2	0.95	1.68	5.22	20.29	80.65	9.27	13.55	32.97	116.02	456.59
PPSI (Ours)	0.077	0.23	0.86	4.35	17.91	2.71	3.21	10.46	31.62	136.49

Table 7. Communication in MB of PPSI and CGS.

$m = n$	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
CGS-PSM1	24.33	99.48	397.65	1700.82	6824.49
CGS-PSM2	17.24	68.9	273.3	1155.7	4637.68
PPSI (Ours)	5.21	20.65	81.31	326.14	1303.99

7.2 Experimental Environment

All the following experiments are conducted on virtual Linux machines running with AMD Ryzen 5 3600 3:60 GHz CPU and 16 GB of memory. All our programs are implemented in C++. And we ran our protocols in two network settings. The bandwidth between the two virtual Linux machines was about 10 GBps (LAN setting) and 100 Mbps (WAN setting), respectively. The round-trip time was about 0.02 ms (LAN setting) and 80 ms (WAN setting), respectively. The network setting is simulated based on the Cheetah framework [1, 16]. Our implementation is built on top of the open-source Cheetah framework [1] provided by the authors of [16], the source code of CM20 [4], a library for private set intersection [35], and the EMP toolkit [38]. Besides, we evaluate the work [3] under the same environment as ours to show the efficiency of our protocols.

7.3 Evaluation for the Applications of PPSI

In this section, we will breakdown PPSI into individual components and give both theoretical and experimental analysis when applying our PPSI protocol in a scenario as depicted in Sect. 5.5. Then, we present the performance of our PM protocol and BatchComp protocol based on different network settings and OT protocols. Next, we combine those components together and evaluate the whole process.

In our first application, we assume the predicate \mathbf{P} is a secure comparison protocol. We evaluate the running times of our PPSI based on different PSI protocols (KKRT [19] and CM20 [5]) in both LAN/WAN settings. we breakdown all individual components and present the execution times of the application of PPSI as shown in Table 4. Besides, we not only perform our PPSI for equal input sizes upto 2^{22} items, but also the unequal sizes. In **Phase 1**, \mathcal{C} and \mathcal{S} will

align their items. We present the time for alignment with different input sizes. Then, in **Phase 2**, \mathcal{C} and \mathcal{S} perform a secure comparison in this application, which performs our BatchComp protocol. Besides, \mathcal{C} and \mathcal{S} use the results from BatchComp to perform the PM protocol. Because \mathcal{C} and \mathcal{S} need to execute b times of secure comparison and PM protocols, and b is only related to the size of n , the running times of this phase are the same when n keeps unchanged. As we can see, if we use CM20 protocol in **Phase 3** of PPSI, it has less communication cost. In LAN setting, it can achieve better efficiency if we choose KKRT protocol as a component of PPSI.

Next, we can combine all individual components together and present the overall execution times of application 2 of PPSI as shown in Table 5. Similarly, we evaluate the running times of our PPSI based on KKRT and CM20 in both LAN/WAN settings. In the experiment of application 1, we use the VOLE-style OT to achieve BatchComp and PM protocols, which have better performance. However, we want to compare our PPSI with CGS [3], which is built based on IKNP-style OT. It would be unfair to use the VOLE-style OT to build our PPSI and compare it with CGS. Therefore, in this application, we evaluate the running time and communication cost of PPSI based on IKNP-style OT. Then, we show the comparison of PPSI and CGS in Sect. 7.5.

Performance of PM Protocol. In PPSI, the number of times PM is run is related to the number of bins of the cuckoo hash tables b in our application. Therefore, the PM protocol is run $b = 1.27n$ times when $n = 2^{14}, 2^{16}, \dots, 2^{22}$. In Table 1 and Table 2, we evaluate the performance of our PM protocol based on IKNP-style OT and VOLE-style OT protocols. As we can see, the VOLE-style OT based PM protocol has reduced communication costs and is much more efficient than the IKNP-style OT based PM protocol.

Performance of BatchComp Protocol. For the BatchComp protocol, we show the results in Table 3. First, we compare BatchComp (IKNP-style OT based) with the comparison protocol in CryptFlow2 in the LAN setting. As we can see, our protocol is about $2.1x$ better than CryptFlow2 in both communication cost and running time. It is easy to learn that our BatchComp protocol is mainly relied on the OT protocol, therefore, except using IKNP-style OT protocol as a main building block of BatchComp, we also implement it based on VOLE-style OT protocol, which only has 1-bit communication cost for an instance of ROT protocol. And this VOLE-style OT based BatchComp protocol is about 1 order of magnitude efficient than IKNP-style OT based in terms of communication.

7.4 Theoretical Analyses

As described in Sect. 5.5, in **Phase 1**, \mathcal{S} performs cuckoo hashing to map n items to a table with b bins and $b = 1.27n$. Then, \mathcal{C} uses simple hashing to get hash tables with b bins. In **Phase 2**, \mathcal{C} and \mathcal{S} performing our BatchComp protocol for b times comparison instead of b instances of the secure comparison protocol in CryptFlow2. The communication cost of b instances of $\binom{K}{1}$ -OT is reduced from

$b(2\lambda + K)$ bits to $(2\lambda + bK)$ bits, for $\lambda = 128$. And the communication cost for \mathcal{F}_{AND} in step 14–22 in Algorithm 5.3 is $b(\lambda + 20)\lceil \log q \rceil + b(2\lambda + 22)(q - 1 - \lceil \log q \rceil)$ bits. Therefore, the communication for performing BatchComp in **Phase 2** is $2\lambda + bK + b(\lambda + 20)\lceil \log q \rceil + b(2\lambda + 22)(q - 1 - \lceil \log q \rceil)$ bits, where $q = 8$ in our application of PPSI. Then, the communication cost for performing b times of PM protocol is $b\lambda$ bits.

7.5 Performance Comparison with CGS

In Table 6 and Table 7, we compare PPSI with CGS [3], one of the state-of-art circuit-PSI protocols. CGS considers the situation that both parties have equal sizes of input sets, so we also perform experiments when $m = n$. There are two protocols proposed in CGS (CGS-PSM1 & CGS-PSM2). We use those two protocol in CGS to achieve the same functionality as shown in **Application 2**, and compare our PPSI protocol with those two protocols in CGS. And the running time and communication cost of PPSI shown in those two tables are IKNP-style OT based. Table 6 shows the running time of PPSI and CGS in the LAN and WAN settings. Under the LAN setting, our protocol is around 5–10x faster than CGS. And for the WAN setting, the end-to-end running time of PPSI is up to 3.9x faster. And Table 7 presents the communication cost of both protocols. Our communication cost is about 3.1x than CGS. Overall, our protocol outperforms CGS in all network settings and different sizes.

Specifically, if a different predicate \mathbf{P} is applied to PPSI and CGS, we only need to trivially substitute the running time of predicate computation in the final results to evaluate the performance. Therefore, the predicate computation would not affect the comparison results. As we can see, our PPSI protocol out-performance CGS in terms of running time and communication costs in different network settings.

8 Conclusion

We presented a two-party efficient PPSI protocol, which has substantial savings of computation cost and communication cost compared to circuit-PSI protocols. PPSI protocol is desirable in many real-world applications. According to different network settings, we have offered different methods to achieve the best performance of PPSI. As suggested in [3, 16], we also tested PPSI based on the VOLE-style OT protocol to reduce communication costs. Based on performance analysis, we believe PPSI has a more practical use. For future work, we will try to build a PPSI protocol that is secure against malicious adversaries. One of the main challenges would be ensuring correctness when the adversaries become malicious.

Acknowledgments. This work was supported by Major Program of Guangdong Basic and Applied Research Project under Grant No. 2019B030302008, National Natural Science Foundation of China under Grant Nos. 61825203, U22B2028 and

62072132, National Key Research and Development Plan of China under Grant No. 2020YFB1005600, Guangdong Provincial Science and Technology Project under Grant No. 2021A0505030033, Science and Technology Major Project of Tibetan Autonomous Region of China under Grant No. XZ202201ZD0006G, National Joint Engineering Research Center of Network Security Detection and Protection Technology, Guangdong Key Laboratory of Data Security and Privacy-Preserving, and Guangdong Hong Kong Joint Laboratory for Data Security and Privacy Protection. We would also thank the anonymous reviewers for their valuable comments.

Appendix 1: Secure Comparison

To compare $x \in \{0, 1\}^\ell$ provided by \mathcal{C} and $y \in \{0, 1\}^\ell$ provided by \mathcal{S} , the secure comparison protocol performs the following four stages:

Splitting stage: \mathcal{C} and \mathcal{S} split their inputs x and y equally into q parts, and q is a power of 2. Each part has k bits, and assume k divides ℓ . Let K be a parameter and $K = 2^k$. That is $x = x^{q-1} || \dots || x^0$ and $y = y^{q-1} || \dots || y^0$, where $x^t, y^t \in \{0, 1\}^k, t \in [0, q - 1]$.

Masking stage: For each part $t \in [0, q - 1]$, \mathcal{C} prepares $\langle \text{lt}_0^t \rangle_{\mathcal{C}}^B, \langle \text{eq}_0^t \rangle_{\mathcal{C}}^B \xleftarrow{\$} \{0, 1\}$. For all $u \in [0, K - 1]$, \mathcal{C} sets $s_u^t = \langle \text{lt}_0^t \rangle_{\mathcal{C}}^B \oplus 1\{x^t < u\}$ and $v_u^t = \langle \text{eq}_0^t \rangle_{\mathcal{C}}^B \oplus 1\{x^t = u\}$.

Choosing stage: \mathcal{C} and \mathcal{S} invoke an instance of $\binom{K}{1}$ -OT $_\ell$ where \mathcal{C} inputs $\{s_u^t\}_{u \in [0, K-1]}$ and \mathcal{S} inputs the choose bit y^t . Then \mathcal{S} gets the output $\langle \text{lt}_0^t \rangle_{\mathcal{S}}^B$. Similarly, \mathcal{C} and \mathcal{S} invoke another instance of $\binom{K}{1}$ -OT $_\ell$ where \mathcal{C} inputs $\{v_u^t\}_{u \in [0, K-1]}$ and \mathcal{S} inputs the choose bit y^t . Then \mathcal{S} gets the output $\langle \text{eq}_0^t \rangle_{\mathcal{S}}^B$.

Merging stage: \mathcal{C} and \mathcal{S} recursively compute the shares they have using the idea of Equation (3). For $i \in [1, \log q]$ and $t \in [1, (q/(2^i) - 1)]$, \mathcal{C} and \mathcal{S} invoke F_{AND} to compute $\langle \text{lt}_i^t \rangle_{\mathcal{C}}^B = \langle \text{lt}_{i-1}^{2t} \rangle_{\mathcal{C}}^B \wedge \langle \text{eq}_{i-1}^{2t+1} \rangle_{\mathcal{C}}^B \oplus \langle \text{lt}_{i-1}^{2t+1} \rangle_{\mathcal{C}}^B$ and $\langle \text{lt}_i^t \rangle_{\mathcal{S}}^B = \langle \text{lt}_{i-1}^{2t} \rangle_{\mathcal{S}}^B \wedge \langle \text{eq}_{i-1}^{2t+1} \rangle_{\mathcal{S}}^B \oplus \langle \text{lt}_{i-1}^{2t+1} \rangle_{\mathcal{S}}^B$. Then \mathcal{C} and \mathcal{S} can compute $\langle \text{eq}_i^t \rangle_{\mathcal{C}}^B = \langle \text{lt}_{i-1}^{2t} \rangle_{\mathcal{C}}^B \wedge \langle \text{eq}_{i-1}^{2t+1} \rangle_{\mathcal{C}}^B$ and $\langle \text{eq}_i^t \rangle_{\mathcal{S}}^B = \langle \text{lt}_{i-1}^{2t} \rangle_{\mathcal{S}}^B \wedge \langle \text{eq}_{i-1}^{2t+1} \rangle_{\mathcal{S}}^B$. Final, \mathcal{C} gets $\langle \text{lt}_i^0 \rangle_{\mathcal{C}}^B$ and \mathcal{S} gets $\langle \text{lt}_i^0 \rangle_{\mathcal{S}}^B$. After the above four stages, \mathcal{C} and \mathcal{S} get the boolean shares of the comparison result of x and y .

Appendix 2: Security Proof

The proof of Theorem 1 is as follows.

Proof. As shown in Algorithm 1, for \mathcal{C} , the view during the protocol execution will be $\mathbf{view}_{\mathcal{C}} = (\langle p^* \rangle_{\mathcal{C}}^B, z)$. Since z is generated by ROT protocol and is a random value, according to Lemma 1, it is trivial to see that all values of \mathcal{C} 's view are uniformly random. \mathcal{C} outputs nothing during this protocol. Therefore, $\mathbf{view}_{\mathcal{C}}$ and $\mathbf{output}_{\mathcal{C}}$ can be simulated by a simulator $\text{Sim}_{\mathcal{C}}$. Then $\text{Sim}_{\mathcal{C}}$ can generate a view for the adversary $\text{Adv}_{\mathcal{C}}$, who can not distinguish the generated view from its real view.

For \mathcal{S} , the view during the protocol execution will be $\mathbf{view}_{\mathcal{S}} = ((p^*)_{\mathcal{S}}^B)$. Then \mathcal{S} outputs R_0, R_1 during this protocol. Since R_0, R_1 is generated by ROT protocol and are random values, according to Lemma 1, all values of \mathcal{S} 's view are uniformly random. Therefore, $\mathbf{view}_{\mathcal{S}}$ and $\mathbf{output}_{\mathcal{S}}$ can be simulated by a simulator $Sim_{\mathcal{S}}$. Then $Sim_{\mathcal{S}}$ can generate a view for the adversary $Adv_{\mathcal{S}}$, who can not distinguish the generated view from its real view.

The proof of Theorem 2 is as follows.

Proof. As shown in Algorithm 5.3, for \mathcal{C} , the view in the protocol execution will be $\mathbf{view}_{\mathcal{C}} = (\alpha, \{s_{1,\alpha^t}^t || \dots || s_{n,\alpha^t}^t\}, \{v_{1,\alpha^t}^t || \dots || v_{n,\alpha^t}^t\})$. Since $\{s_{1,\alpha^t}^t || \dots || s_{n,\alpha^t}^t\}, \{v_{1,\alpha^t}^t || \dots || v_{n,\alpha^t}^t\}$ are generated from the random elements (step 7–8 in Algorithm 5.3), according to Lemma 1, it is trivial to see that all values of \mathcal{C} 's view are uniformly random. The output of \mathcal{C} is $\mathbf{output}_{\mathcal{C}} = \langle 1\{\alpha < p_i\} \rangle_{\mathcal{C}}$, which is generated from the random values $\{s_{1,\alpha^t}^t || \dots || s_{n,\alpha^t}^t\}, \{v_{1,\alpha^t}^t || \dots || v_{n,\alpha^t}^t\}$ (step 14–20 in Algorithm 5.3). Therefore, $\mathbf{view}_{\mathcal{C}}$ and $\mathbf{output}_{\mathcal{C}}$ can be simulated by a simulator $Sim_{\mathcal{C}}$. Then $Sim_{\mathcal{C}}$ can generate a view for the adversary $Adv_{\mathcal{C}}$, who can not distinguish the generated view from its real view.

For \mathcal{S} , the view in the protocol execution will be $\mathbf{view}_{\mathcal{S}} = (\mathbf{p}, \langle lt_{0,j}^t \rangle, \langle eq_{0,j}^t \rangle)$. Since $\{\langle lt_{0,j}^t \rangle, \langle eq_{0,j}^t \rangle\}$ are generated from the random elements (step 5 in Algorithm 5.3), all values of \mathcal{S} 's view are uniformly random. The output of \mathcal{S} is $\mathbf{output}_{\mathcal{S}} = \langle 1\{\alpha < p_i\} \rangle_{\mathcal{S}}$, which is generated from the random values $\{\langle lt_{0,j}^t \rangle, \langle eq_{0,j}^t \rangle\}$ (step 6–8 in Algorithm 5.3). Therefore, $\mathbf{view}_{\mathcal{S}}$ and $\mathbf{output}_{\mathcal{S}}$ can be simulated by a simulator $Sim_{\mathcal{S}}$. Then $Sim_{\mathcal{S}}$ can generate a view for the adversary $Adv_{\mathcal{S}}$, who can not distinguish the generated view from its real view.

References


1. Alibaba-Gemini-Lab: Opencheetah (2022). <https://github.com/Alibaba-Gemini-Lab/OpenCheetah>
2. Bogdanov, D., Naitsoo, M., Toft, T., Willemsen, J.: High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Secur.* **11**(6), 403–418 (2012)
3. Chandran, N., Gupta, D., Shah, A.: Circuit-psi with linear complexity via relaxed batch OPRF. *Proc. Privacy Enhanc. Technol.* **1**, 353–372 (2022)
4. Chase, M., Miao, P.: OPRF-PSI (2020). <https://github.com/peihanmiao/OPRF-PSI>
5. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Micciancio, D., Ristenpart, T. (eds.) *CRYPTO 2020*. LNCS, vol. 12172, pp. 34–63. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_2
6. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1223–1237. ACM, Los Angeles, CA, USA (2018)
7. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1243–1255. ACM, New York, NY, United States (2017)

8. Cong, K., et al.: Labeled PSI from homomorphic encryption with reduced computation and communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 1135–1150. ACM, New York, NY, United States (2021)
9. Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
10. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: ACM SIGSAC Conference on Computer and Communications Security, pp. 789–800. ACM, Berlin, Germany (2013)
11. EdalatNejad, K., Raynal, M., Lueks, W., Troncoso, C.: Private set matching protocols. arXiv preprint [arXiv:2206.07009](https://arxiv.org/abs/2206.07009) (2022)
12. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
13. Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S., Singh, J.: Private set operations from oblivious switching. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12711, pp. 591–617. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_21
14. Huang, K., Liu, X., Fu, S., Guo, D., Xu, M.: A lightweight privacy-preserving CNN feature extraction framework for mobile sensing. IEEE Trans. Depend. Secur. Comput. **18**(3), 1441–1455 (2019)
15. Huang, Y., Evans, D., Katz, J.: Private set intersection: are garbled circuits better than custom protocols? In: NDSS, San Diego, California, USA (2012)
16. Huang, Z., Lu, W.J., Hong, C., Ding, J.: Cheetah: lean and fast secure two-party deep neural network inference. Cryptology ePrint Archive (2022)
17. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9
18. Kolesnikov, V., Kumaresan, R.: Improved OT extension for transferring short secrets. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 54–70. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_4
19. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 818–829. ACM, New York, USA (2016)
20. Laur, S., Talviste, R., Willemsen, J.: From oblivious AES to efficient and secure database join in the multiparty setting. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 84–101. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38980-1_6
21. Le, P.H., Ranellucci, S., Gordon, S.D.: Two-party private set intersection with an untrusted third party. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2403–2420. ACM, New York, USA (2019)
22. Lepoint, T., Patel, S., Raykova, M., Seth, K., Trieu, N.: Private join and compute from pir with default. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13091, pp. 605–634. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92075-3_21
23. Lindell, Y.: How to simulate it—a tutorial on the simulation proof technique. Tutor. Found. Cryptogr. 277–346 (2017)
24. Liu, Y., Zhang, X., Wang, L.: Asymmetrical vertical federated learning. arXiv preprint [arXiv:2004.07427](https://arxiv.org/abs/2004.07427) (2020)

25. Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: 1986 IEEE Symposium on Security and Privacy, pp. 134–134. IEEE (1986)
26. Mohassel, P., Rindal, P., Rosulek, M.: Fast database joins and psi for secret shared data. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1271–1287 (2020)
27. Mohassel, P., Rosulek, M., Trieu, N.: Practical privacy-preserving k-means clustering. Cryptology ePrint Archive (2019)
28. Pagh, R., Rodler, F.F.: Cuckoo hashing. *J. Algorithms* **51**(2), 122–144 (2004)
29. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: SpOT-light: lightweight private set intersection from sparse OT extension. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 401–431. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_13
30. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from PaXoS: fast, malicious private set intersection. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 739–767. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_25
31. Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based PSI with linear communication. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 122–153. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_5
32. Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based PSI via cuckoo hashing. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 125–157. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_5
33. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. *ACM Trans. Privacy Secur. (TOPS)* **21**(2), 1–35 (2018)
34. Rathee, D., et al.: Cryptflow2: practical 2-party secure inference. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 325–342. ACM, New York, USA (2020)
35. Rindal, P.: libpsi (2020). <https://github.com/osu-crypto/libPSI>
36. Taassori, M., Shafiee, A., Balasubramonian, R.: Vault: reducing paging overheads in SGX with efficient integrity verification structures. In: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 665–678. ACM, New York, USA (2018)
37. Takeshita, J., Karl, R., Mohammed, A., Striegel, A., Jung, T.: Provably secure contact tracing with conditional private set intersection. In: Garcia-Alfaro, J., Li, S., Poovendran, R., Debar, H., Yung, M. (eds.) SecureComm 2021. LNICST, vol. 398, pp. 352–373. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90019-9_18
38. Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: efficient MultiParty computation toolkit (2016). <https://github.com/emp-toolkit>
39. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: fast extension for correlated OT with small communication. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1607–1626. ACM, New York, USA (2020)
40. Ying, J.H., Cao, S., Poh, G.S., Xu, J., Lim, H.W.: PSI-stats: private set intersection protocols supporting secure statistical functions. In: Ateniese, G., Venturi, D. (eds.) ACNS 2022. LNCS, vol. 13269, pp. 585–604. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-09234-3_29



A Framework for UC Secure Privacy Preserving Biometric Authentication Using Efficient Functional Encryption

Johannes Ernst^(✉)  and Aikaterini Mitrokotsa

University of St. Gallen, St. Gallen, Switzerland
{johannes.ernst,katerina.mitrokotsa}@unisg.ch

Abstract. Despite its popularity, password based authentication is susceptible to various kinds of attacks, such as online or offline dictionary attacks. Employing biometric credentials in the authentication process can strengthen the provided security guarantees, but raises significant privacy concerns. This is mainly due to the inherent variability of biometric readings that prevents us from simply applying a standard hash function to them. In this paper we first propose an ideal functionality for modeling secure, privacy preserving biometric based two-factor authentication in the framework of universal composability (UC). The functionality is of independent interest and can be used to analyze other two-factor authentication protocols. We then present a generic protocol for biometric based two-factor authentication and prove its security (relative to our proposed functionality) in the UC framework. The first factor in our protocol is the possession of a device that stores the required secret keys and the second factor is the user's biometric template. Our construction can be instantiated with function hiding functional encryption, which computes for example the distance of the encrypted templates or the predicate indicating whether the templates are close enough. Our contribution can be split into three parts:

- We model privacy preserving biometric based two-factor authentication as an ideal functionality in the UC framework. To the best of our knowledge, this is the first description of an ideal functionality for biometric based two-factor authentication in the UC framework.
- We propose a general protocol that uses functional encryption and prove that it UC-realizes our ideal functionality.
- We show how to instantiate our framework with efficient, state of the art inner-product functional encryption. This allows the computation of the Euclidean distance, Hamming distance or cosine similarity between encrypted biometric templates. In order to show its practicality, we implemented our protocol and evaluated its performance.

Keywords: cryptographic protocols · biometric authentication · privacy preserving computation · universal composability

1 Introduction

It is long known that password based authentication is not very secure. Users tend to choose bad passwords and frequent leaks of password databases enable offline attacks. Biometric authentication mitigates many of these problems and has better usability, since users do not have to remember their passwords anymore. However, the use of biometric authentication entails new problems. Biometric readings present inherent variability and, thus, common protection mechanisms such as standard hash functions or encryption cannot be employed. Furthermore, the client's biometric template (e.g. face, iris-scan or fingerprint) can be considered privacy sensitive, as it may reveal ethnic origin or even health conditions and, therefore, should remain private. Also, unlike passwords, biometrics cannot be changed, hence, the breach of a biometric database may have more severe consequences than the breach of a password database. Thus, for both privacy and security reasons it is important to achieve reliable authentication, while preventing the server from seeing the client's biometric templates. A general technique to make authentication more secure is two-factor authentication. This incorporates a second factor in the authentication process, so that in addition to the password or biometric the user needs for example a secret key that is stored on a phone or a secure hardware device. This makes an attack more difficult because it requires the attacker to get both the device and the biometric/password.

Homomorphic encryption and general multiparty computation have been used in the past to construct privacy preserving biometric authentication systems. However, using general multiparty computation usually requires multiple rounds of communication, which can reduce efficiency. Approaches using homomorphic encryption (e.g. [19]) have the problem that some part of the server needs to know the secret key to be able to decrypt the authentication decision. This also means that, when the entire server infrastructure is compromised, the attacker learns both the secret key and the encrypted templates and thereby the cleartext templates. Functional encryption (FE) is similar to homomorphic encryption in that it allows computing on encrypted data. The important advantage of FE in the biometric authentication setting is that it allows the server to learn the result of the computation in cleartext without having access to the secret key that allows decryption of the templates. Depending on the underlying FE scheme, the server only learns the distance of the biometric templates, or only the fact whether this distance is below a certain threshold. This makes FE well suited for the use case of biometric authentication.

Function hiding inner-product functional encryption (fh-IPFE) essentially allows computing the inner-product of two encrypted vectors. Therefore, fh-IPFE and its restricted and more efficient variant of single-key fh-IPFE has been used in the past to construct privacy preserving biometric authentication and identification systems [5, 7, 15, 18]. Since one of the biometric templates is encoded as the function, the function hiding property is important, as otherwise this template would not be hidden. Because fh-IPFE only exists in the secret key setting and this key needs to be stored somewhere, it is very natural to consider fh-IPFE in the setting of two-factor authentication, where the first factor is the secret key stored on the device and the second factor is the user's biometric.

However, the security of these constructions has not been studied in the context of more complex systems in which multiple users have accounts on multiple servers and the authentication protocol is executed in conjunction with other protocols. For example a user could try to authenticate towards two different servers at the same time with very similar or identical biometric templates. Even if both servers are corrupted they should learn no more than the output of the authentication protocol. It is also important that an attacker, who is in control of the network, can neither learn anything about the biometric templates, nor impersonate a legitimate user. Furthermore, a malicious server should not be able to use the authentication message of a user to impersonate that user to another server.

Let us illustrate the danger of too restrictive security models with a concrete example. For their biometric authentication system the authors of [7] only consider the privacy of the biometric templates in a setting with a single client and a single server. However, the model does not guarantee security, i.e. it does not guarantee that it is hard to impersonate an honest client. Concretely, in the security definition the adversary gets access to one oracle for each the enrolment phase and the authentication phase. In the real world these oracles call the actual IPFE scheme, whereas in the ideal world a simulator has to produce the answers without knowing the biometric template. This guarantees that an attacker cannot learn anything about the biometric templates from the ciphertexts. However, this does not guarantee that it is hard to impersonate an honest client. In fact, an attacker can send a random vector as ciphertext, which causes the output to be a random number. With their first parameter-set this is a random number in $\mathbb{Z}_{2^{20}}$. When the threshold for the biometric authentication system is e.g. $\tau = 32$, then the probability that a random number is below τ is 2^{-15} . This is a too high chance to impersonate a legitimate user for an attacker who does neither know the secret key nor the user's biometric. This example illustrates why it is important to model both privacy *and* security in complex environments.

Contribution: In order to fill this gap, we model biometric authentication as an ideal functionality in the framework of universal composability (UC) [6]. This guarantees that the resulting protocols remain secure even in complex systems and in the presence of powerful attackers. Our ideal functionality can also serve as basis for the analysis of new protocols for two-factor authentication. In the second step we propose a biometric based two-factor authentication protocol that generically uses (single-key) function hiding functional encryption (fh-FE) and signatures and formally prove that it realizes our ideal functionality. The first factor is the secret key that is stored in secure hardware and the second factor is the biometric template. Next, we show how to instantiate our framework with (single-key) fh-IPFE schemes so that it can compute either the Euclidean distance, Hamming distance, or cosine similarity on encrypted biometric templates. We stress that our framework can also be instantiated in a way (e.g. with techniques of [5]) that does not leak the distance of the templates to the server, but only outputs the yes-no authentication decision. Finally, we describe the proof of concept implementation of our protocol and present the results of the performance tests. Thereby, we get a secure, privacy preserving and composable

biometric authentication protocol. Furthermore, the protocol is practically efficient and both enrolment and authentication only require a single message from the client to the server. Our contribution can be summarized as follows:

- We provide a UC formalization of (two-factor) biometric authentication.
- We propose a general protocol/framework that realizes our ideal functionality, which can be instantiated with fh-FE and signatures.
- We provide a concrete instantiation with fh-IPFE for computing the Euclidean distance, Hamming distance or cosine similarity and a proof of concept implementation.

1.1 Related Work

Universally Composable Biometric Authentication: Universally composable biometric authentication has received some attention in the literature [1, 2, 10, 11]. Dupont et al. [10] considers fuzzy (symmetric) password authenticated key exchange, where the server also has to know the password in the clear. This problem is solved in [11] by moving to the asymmetric setting. Their protocol can be used for biometric authentication with binary vectors and the Hamming distance. Asymmetric password authenticated key exchange is a stronger notion of authentication than ours, since it considers mutual authentication, whereas in our case only the client authenticates to the server. However, it is usually less efficient, as Erwig et al. [11] note that “. . . going beyond password sizes of, say, 40 bits does not seem feasible.”, where e.g. iris templates for the Hamming distance are usually more than 1000 bits long (see e.g. [4, 8]). Agrawal et al. [1] construct a biometric authentication system in which the reference template is secret shared among three client devices. However, having three connected devices whenever one wants to authenticate is not always practical. Agrawal et al. [2] consider the scenario in which a client wants to get access to a certain area and the client’s device, an external terminal and service provider interact in order to perform the enrolment and the biometric matching. This is a different setting than ours.

Function Hiding Inner-Product Functional Encryption for Biometrics: Function hiding IPFE has already been used for biometric authentication [7] and identification [5, 15, 18], however their security models do not take composability into account. Composability is an important property when a protocols runs in conjunction with other protocols in a complex environment like the internet. Both [15, 18] only rely on the security definition of IPFE but have no authentication or identification related security model. The security model of [5] does take biometrics into account, but is specific to searchable encryption. The work most closely related to ours is [7], because it is the only one that directly considers biometric *authentication*. Their security model is for biometric authentication, however it only considers one user and one server and does not guarantee that it is hard to impersonate another user. Both [7, 18] have proposed constructions for fh-IPFE. Contrary to existing work, we propose for the first time a general protocol for privacy-preserving biometric based two-factor authentication that provides UC

security guarantees and can be instantiated using any suitable fh-FE scheme including the ones proposed by [7, 18].

Other Biometric Authentication Protocols: There are many other protocols for privacy preserving biometric authentication, such as [3, 13, 14, 24, 25]. The authors of [13, 24] consider biometric authentication for secure messaging and text search, respectively, which is a different setting than ours. The authors of [3, 14, 25] consider the classical setting of a client authenticating to a server. The disadvantage of [3, 14] is that the protocols need two non-colluding servers and that they are not secure against fully malicious attackers. The authors of [25], only consider the privacy of the client’s biometrics, but not security against e.g. impersonations. None of them takes security under general composition into account.

Jarecki et al. [17] propose protocols for secure, password based two-factor authentication, where the user has a password and additionally owns a “crypto-capable device”. They provide a very detailed security analysis of their protocol in a game base setting. The main difference to our setting is that we use the UC framework and that we work with biometrics instead of passwords.

2 Preliminaries

Notation: We write $[n]$ for $\{1, \dots, n\}$. We use boldface letters such as \mathbf{v} for vectors and we write v_i for the i -th entry of \mathbf{v} . With $\|\mathbf{v}\|$ we denote the L2 norm of \mathbf{v} . We write \mathcal{C} for a client, \mathcal{S} for a server, \mathcal{A} for the adversary, Sim for the simulator and \mathcal{Z} for the environment. With *reference* template we mean the biometric template used for enrolment and with *probe* template we mean the template used for authentication. We usually denote them as \mathbf{b} and \mathbf{b}' .

2.1 (Secret Key) Function Hiding Functional Encryption

Definition 1. *A secret key functional encryption scheme for a set of functions \mathcal{F} , which map from \mathcal{X} to \mathcal{Y} consists of the following four algorithms:*

- $\text{FE.Setup}(1^\lambda)$ outputs public parameters \mathbf{pp} and the master secret key msk
- $\text{FE.KeyGen}(\text{msk}, f \in \mathcal{F})$ outputs a functional decryption key sk_f
- $\text{FE.Enc}(\text{msk}, x \in \mathcal{X})$ outputs a ciphertext c_x
- $\text{FE.Dec}(\text{sk}_f, c_x)$ outputs a value $y \in \mathcal{Y}$

We assume that \mathbf{pp} is given as implicit input to all other FE algorithms.

Correctness: A functional encryption scheme FE is correct if $\forall \text{msk} \leftarrow \text{FE.Setup}(1^\lambda), \forall f \in \mathcal{F}, \forall x \in \mathcal{X}$ it holds that $\text{FE.Dec}(\text{FE.KeyGen}(\text{msk}, f), \text{FE.Enc}(\text{msk}, x)) = f(x)$.

In the special case of inner-product functional encryption (IPFE), which we use for our instantiation in Sect. 5, the inputs are $\mathcal{X} := \mathbb{Z}_q^n \setminus \{0^n\}$ and $\mathcal{F} := \{f_y : y \in \mathbb{Z}_q^n \setminus \{0^n\}\}$, where $f_y(x) := \langle y, x \rangle$.

Definition 2. For $b \in \{0, 1\}$ we define $\text{Exp}_{\mathcal{A}}^b(\lambda)$ as the following experiment: The experiment generates $(\text{pp}, \text{msk}) \leftarrow \text{FE.Setup}(\lambda)$ and gives pp to \mathcal{A} . The experiment then answers the following types of oracle queries from \mathcal{A} :

- $\text{QKeyGen}(f_0, f_1)$: The experiment returns $\text{sk} \leftarrow \text{FE.KeyGen}(\text{msk}, f_b)$.
- $\text{QEnc}(x_0, x_1)$: The experiment returns $c \leftarrow \text{FE.Enc}(\text{msk}, x_b)$.

When \mathcal{A} outputs a guess bit b' , $\text{Exp}_{\mathcal{A}}^b$ outputs the same b' .

Definition 3. (Admissible adversaries) For an adversary \mathcal{A} , let $(f_0^1, f_1^1), \dots, (f_0^{Q_K}, f_1^{Q_K})$ and $(x_0^1, x_1^1), \dots, (x_0^{Q_E}, x_1^{Q_E})$ be the QKeyGen and QEnc queries. We say that an adversary is admissible if $\forall i \in [Q_K], \forall j \in [Q_E]: f_0^i(x_0^j) = f_1^i(x_1^j)$.

Considering only admissible adversaries prevents \mathcal{A} from trivially winning the game by querying functions and inputs with different output values. We say that an FE-scheme FE is fh-IND-secure, if for all admissible PPT adversaries \mathcal{A} there is a negligible function $\text{negl}(\lambda)$ such that for all large enough λ

$$|\Pr[\text{Exp}_{\mathcal{A}}^0(\lambda) = 1] - \Pr[\text{Exp}_{\mathcal{A}}^1(\lambda) = 1]| \leq \text{negl}(\lambda).$$

We say that an FE-scheme FE is *single-key* fh-IND-secure if the above holds for the modified version of the experiment where only the first QKeyGen query is answered, i.e. \mathcal{A} only gets a single key. The function hiding property of the definition is given by the fact that \mathcal{A} does not know whether they got a secret key for f_0 or f_1 .

3 Modeling Biometric Authentication in the UC Framework

In this section we describe our approach to modeling biometric based two-factor authentication in the UC framework. We start by describing the threat model and proceed by motivating and stating our ideal functionality. We then explain the meaning of the functionality's different interfaces. Furthermore, we describe how we model message transfer and corruptions in the real world. We conclude by discussing some design decisions and the security and privacy guarantees that our functionality provides.

3.1 Threat Model

In this section we describe the threat model we consider. We will discuss the attacker's capabilities in more detail in Sect. 3.2, where we explain the different interfaces of the ideal functionality. We expect our protocol to be used for authentication over the internet. Communication is supposed to be done via TLS, which means that clients can send messages to a server and be sure that only the server can read them. On the other hand, the server does not know who the sender of a message is. We assume that the attacker controls the network.

So the attacker notices when a message is sent. Although the attacker cannot read the messages sent via TLS, they can delay or block them or inject their own messages.

The attacker can corrupt both arbitrary clients and the server and then control their actions in arbitrary ways, i.e. we consider malicious corruptions. For simplicity we only consider static corruptions of the server which means, that the server is corrupted either the entire time or not corrupted at all. Clients can be corrupted adaptively, i.e. the attacker can decide at any point in time to corrupt an arbitrary client. However, looking ahead, we will assume that the clients' keys are stored in secure hardware. So when the attacker corrupts a client, they will only gain control over that client's actions and gain blackbox access to the secure hardware. This means, they can give to the secure hardware instructions to enrol or authenticate, but they do not learn the internal state of the secure hardware. The use of secure hardware is also the reason why the attacker does not immediately learn the user's biometric reference template, even if they corrupt both the server and the client. Nevertheless, in the case that the adversary corrupts both the client *and* the server, they may be able to extract the biometric reference template (cf. Section 3.2 for an explanation of when this is possible).

3.2 The Ideal Functionality

We will first describe a simple ideal functionality in Fig. 1 for two-factor authentication with biometrics as a second factor. Then we explain the changes that we applied to this simple functionality in order to get to the actual two-factor authentication functionality $\mathcal{F}_{2FA}^{\text{out}}$ in Fig. 2. Both functionalities are parameterized with the $\text{out}(\cdot, \cdot)$ function, which will be part of our framework. It models both the output that the server gets from the protocol and the information that is leaked about the biometric templates. For example the out function could return the distance of the biometric templates, which would then mean that the server learns this distance. Alternatively the out function could only return one bit, indicating whether the biometric templates are close enough.

- On $(\text{ENROL}, \text{sid}, \text{uid}, \text{b})$ from \mathcal{C}
 - if there is *no* record $(\cdot, \cdot, \mathcal{C})$:
 - * store $(\text{uid}, \text{b}, \mathcal{C})$ and send $(\text{ENROL}, \text{sid}, \text{uid})$ to \mathcal{S}
- On $(\text{AUTH}, \text{sid}, \text{uid}, \text{b}')$ from \mathcal{C} :
 - if there is a record $(\text{uid}, \text{b}, \mathcal{C})$:
 - * send $(\text{AUTH}, \text{sid}, \text{uid}, \text{out}(\text{b}, \text{b}'))$ to \mathcal{S}
 - else send $(\text{AUTH-FAIL}, \text{sid})$ to \mathcal{S}

Fig. 1. The code of the simplified ideal functionality.

We want to capture the setting of two-factor authentication where the first factor is a secret key stored on the user's phone or a dedicated hardware device. The second factor is the user's biometric. In our model the client \mathcal{C} is the device

on which the secret key is stored and the user is the actual person. For enrolment the client sends the user-id uid , such as an email address, and the biometric reference template \mathbf{b} to the ideal functionality. If that client is not yet enrolled, the ideal functionality stores an internal record and notifies the server that a client with uid has enrolled. Note that the server gets no information about \mathbf{b} .

For the authentication, \mathcal{C} again sends the user-id and a fresh biometric template \mathbf{b}' to the ideal functionality. If and only if the same client has previously enrolled with the same user-id, the ideal functionality gives $\text{out}(\mathbf{b}, \mathbf{b}')$ to \mathcal{S} . Depending on the underlying FE scheme this can be the distance of the templates or the yes-no authentication decision. Note that apart from $\text{out}(\mathbf{b}, \mathbf{b}')$ the server learns nothing about the biometric templates. The fact that the ideal functionality checks that the identity \mathcal{C} of the client in the authentication phase is the same as the identity of the client in the enrolment phase represents the first factor, i.e. possession of the secret key. Because in the UC framework clients cannot fake their identity, a client cannot authenticate with the uid of another client. The second factor, i.e. the biometric authentication is represented by the ideal functionality storing the biometric templates and giving $\text{out}(\mathbf{b}, \mathbf{b}')$ to the server.

This simple functionality, however, does not capture all actions that an adversary can take in the real world. Therefore, we added several interfaces in order to model the adversary's capabilities. Also we exchanged the uid by a randomly chosen rid to avoid that two clients enrol with the same identifier. The resulting functionality $\mathcal{F}_{2\text{FA}}^{\text{out}}$ is shown in Figure Fig. 2.

Meaning of the interfaces of $\mathcal{F}_{2\text{FA}}^{\text{out}}$: Here we explain the meaning of all the interfaces of $\mathcal{F}_{2\text{FA}}^{\text{out}}$ and their connection to reality.

The (ENROL, sid , ssid , \mathbf{b}) and (AUTH, sid , ssid , \mathbf{b}) interface of a client \mathcal{C} : These are the interfaces that a client would use for normal enrolment and authentication. All other interfaces are there to model the adversary's capabilities. We let the environment \mathcal{Z} choose the biometric template and the time of enrolment/authentication, because in practice we do not have control over the biometric template or the timing. In UC-PAKE for example the environment also chooses the credential i.e. the password and the time of enrolment/authentication (cf. [16]). Letting the environment choose the biometric templates of the clients and the time (and order) of their enrolments/authentications is in a sense like taking the worst case of possible events in reality.

The ENROK and AUTHOK interfaces: In the real world the adversary can delay or block messages. The ENROK and AUTHOK interfaces model the fact that messages are only delivered when \mathcal{A} explicitly allows this.

The (ENROL, sid , ssid , rid , \mathbf{b}) and (AUTH, sid , ssid , rid , \mathbf{b}') interface of \mathcal{A} : These two interfaces are necessary because the adversary has more control over the records of corrupted clients. Our ideal functionality in Fig. 2 has an ENROL interface which explicitly allows the adversary to enrol malicious clients with arbitrary rid . All clients enrolled in that way are explicitly stored as *adversarially enrolled* clients by $\mathcal{F}_{2\text{FA}}^{\text{out}}$. The adversary can then use its AUTH interface

This functionality interacts with a server \mathcal{S} (specified in the sid) and arbitrary clients. It is parameterized by the function $\text{out}(\cdot, \cdot)$, which determines the output that \mathcal{S} gets.

Enrolment:

- On $(\text{ENROL}, \text{sid}, \text{ssid}, \text{b})$ from \mathcal{C}
 - if there is *no* record $\langle \text{enrol-request}, \cdot, \cdot, \mathcal{C} \rangle$:
 - * store $\langle \text{enrol-request}, \text{ssid}, \text{b}, \mathcal{C} \rangle$
 - * send $(\text{ENROL}, \text{sid}, \text{ssid}, \mathcal{C})$ to \mathcal{A}
- On $(\text{ENROLOK}, \text{sid}, \text{ssid})$ from \mathcal{A}
 - if there is a record $\langle \text{enrol-request}, \text{ssid}, \text{b}, \mathcal{C} \rangle$ and *no* record $\langle \text{enroled}, \mathcal{C}, \cdot, \cdot \rangle$:
 - * sample $\text{rid} \leftarrow_{\mathcal{S}} \{0, 1\}^\lambda$
 - * store $\langle \text{enroled}, \mathcal{C}, \text{rid}, \text{b} \rangle$
 - * if the record $\langle \text{enrol-request}, \text{ssid}, \text{b}, \mathcal{C} \rangle$ is marked **CORRUPTED**:
 - mark the record $\langle \text{enroled}, \mathcal{C}, \text{rid}, \text{b} \rangle$ as **CORRUPTED**
 - * send $(\text{ENROL}, \text{sid}, \text{ssid}, \text{rid})$ to \mathcal{S}
- On $(\text{ENROL}, \text{sid}, \text{ssid}, \text{rid}, \text{b})$ from \mathcal{A} :
 - if there is a record $\langle \text{enroled-adversarial}, \text{rid}, \cdot \rangle$ or $\langle \text{enroled}, \cdot, \text{rid}, \cdot \rangle$:
 - * send $(\text{ENROL}, \text{sid}, \text{ssid}, \perp)$ to \mathcal{S}
 - **else**: store $\langle \text{enroled-adversarial}, \text{rid}, \text{b} \rangle$ and send $(\text{ENROL}, \text{sid}, \text{ssid}, \text{rid})$ to \mathcal{S}

Authentication:

- On $(\text{AUTH}, \text{sid}, \text{ssid}, \text{b}')$ from \mathcal{C} :
 - if there is a record $\langle \text{enrol-request}, \cdot, \cdot, \mathcal{C} \rangle$:
 - * store $\langle \text{auth-request}, \text{ssid}, \text{b}', \mathcal{C} \rangle$
 - * send $(\text{AUTH}, \text{sid}, \text{ssid}, \mathcal{C})$ to \mathcal{A}
- On $(\text{AUTHOK}, \text{sid}, \text{ssid})$ from \mathcal{A} :
 - if there is a record $\langle \text{auth-request}, \text{ssid}, \text{b}', \mathcal{C} \rangle$:
 - * if there is a record $\langle \text{enroled}, \mathcal{C}, \text{rid}, \text{b} \rangle$:
 - delete the record $\langle \text{auth-request}, \text{ssid}, \text{b}', \mathcal{C} \rangle$ and send $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, \text{out}(\text{b}, \text{b}'))$ to \mathcal{S}
 - * **else** delete the record $\langle \text{auth-request}, \text{ssid}, \text{b}', \mathcal{C} \rangle$ and send $(\text{AUTH-FAIL}, \text{sid}, \text{ssid})$ to \mathcal{S}
- On $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, \text{b}')$ from \mathcal{A} :
 - if there is a record $\langle \text{enroled-adversarial}, \text{rid}, \text{b} \rangle$
 - * send $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, \text{out}(\text{b}, \text{b}'))$ to \mathcal{S}
 - **else** send $(\text{AUTH-FAIL}, \text{sid}, \text{ssid})$ to \mathcal{S}

Corruption and impersonation:

- On $(\text{CORRUPT}, \text{sid}, \mathcal{C})$ from \mathcal{A} :
 - if there is a record $\langle \text{enroled}, \mathcal{C}, \text{rid}, \text{b} \rangle$:
 - * mark the record **CORRUPTED**
 - * send $(\text{CORRUPTED}, \text{sid}, \text{rid})$ to \mathcal{A}
 - if there is a record $\langle \text{enrol-request}, \text{ssid}, \cdot, \mathcal{C} \rangle$:
 - * mark the record **CORRUPTED**
 - * send $(\text{CORRUPTED}, \text{sid}, \text{rid})$ to \mathcal{A}
- On $(\text{TRYIMPERSONATE}, \text{sid}, \text{ssid}, \mathcal{C}, \text{b}')$ from \mathcal{A} :
 - if there is a record $\langle \text{enroled}, \mathcal{C}, \text{rid}, \text{b} \rangle$ that is marked **corrupted**:
 - * send $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, \text{out}(\text{b}, \text{b}'))$ to \mathcal{S}

Fig. 2. The code of the ideal functionality $\mathcal{F}_{2\text{FA}}^{\text{out}}$.

to authenticate in the name of any such client. Importantly, the adversary has, through their ENROL and AUTH interfaces, only influence on the records of the adversarially enroled clients. This guarantees that the adversary cannot authenticate in the name of an honest client and, thus, cannot impersonate an honest client by using this interface.

The CORRUPT and TRYIMPERSONATE interfaces: The adversary can use the CORRUPT interface to corrupt an honest client and thereby bypass the first authentication factor. In reality this could mean that the adversary has gained remote access to the device that the client uses. It can also mean that the adversary has stolen the device or secure hardware token and has physical access to it. These two cases are modeled by the same interface, because in both cases the adversary gains the same capabilities, namely to try to impersonate the client with a self-chosen biometric template. This is modeled by the TRYIMPERSONATE interface, where \mathcal{A} can specify a client \mathcal{C} and a biometric template \mathbf{b}' . If \mathcal{C} has been corrupted previously, $\mathcal{F}_{2\text{FA}}^{\text{out}}$ will send $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, \text{out}(\mathbf{b}, \mathbf{b}'))$ to the server \mathcal{S} . Thus, essentially the impersonation attempt will only succeed if \mathcal{A} knows a biometric template \mathbf{b}' that is close enough to the reference template \mathbf{b} and thereby bypasses the second authentication factor. This meaningfully models reality where the adversary can only successfully impersonate an honest client if they both have access to the device and know a matching biometric template.

Modeling Corruptions in the Real World: The code in Fig. 3 models the client’s behavior upon corruption. This code is not part of the protocol that would be executed in reality, but is added to the client in the real world experiment in the UC framework in order to properly model client corruptions. In the language of the UC framework, this code describes the behavior of the client’s *shell*, whereas the *body* contains the actual protocol code. Later in our protocol we assume the use of secure hardware for storing key material and computing the enrolment and authentication messages, which is also reflected in the code in Fig. 3. Namely, the adversary does not get the internal state of the client, but only blackbox access to the secure hardware. This means that \mathcal{A} can try to impersonate a client by giving a biometric template \mathbf{b}' to the secure hardware and forwarding the message to the server. We distinguish the *host* from the secure hardware. The host would usually be the software on a smartphone or laptop and the secure hardware would be a special chip in the phone or laptop, or a USB-stick like hardware token.

When one wants to instantiate our model without the use of secure hardware then it is necessary to adapt the code in Fig. 3 to return the entire local state of the client and then exactly follow the instructions from \mathcal{A} .

Modeling Message Transfer in the Real World: Messages in our framework will be sent via the $\mathcal{F}'_{\text{SMT}}$ functionality (Fig. 4), which is an adaption of \mathcal{F}_{SMT} from [6]. It is meant to model TLS connections where the server is authenticated via a certificate but the client is not authenticated. This is modeled by the fact that $\mathcal{F}'_{\text{SMT}}$, as opposed to \mathcal{F}_{SMT} , does not give the client’s identity \mathcal{C} to the server. However, the client can be sure that only the server can read the message. It is important to model this, because giving the client’s identity \mathcal{C} to the server would make further authentication unnecessary. Simply put, we will use $\mathcal{F}'_{\text{SMT}}$ to capture the client’s instruction “send m to \mathcal{S} over the internet via TLS”.

This code describes the behavior (of the shell) of a client \mathcal{C} upon corruption.

- On (CORRUPT, sid) from \mathcal{A}
 - if \mathcal{C} is enrolled (i.e. (ENROL, ·, ·)) has been called before:
 - * set flag `corrupted`
 - * from now on ignore every input from \mathcal{Z} and only take instructions from \mathcal{A}
 - * send (CORRUPTED, sid) to \mathcal{A}
- On (TRYIMPERSONATE, sid, ssid, b') from \mathcal{A}
 - if flag `corrupted` is set
 - * give (AUTH, sid, ssid, b') to the secure hardware in the name of the host
 - * when getting an output m from the secure hardware, send m to \mathcal{A}

Fig. 3. The code modeling client behavior upon corruption.

- On (SEND, sid, \mathcal{S} , m) from a client \mathcal{C}
 - send (SENT, sid, \mathcal{C} , \mathcal{S} , $l(m)$) to \mathcal{A}
- On (OK, sid) from \mathcal{A} :
 - If not yet generated output then send (SENT, sid, m) to server \mathcal{S}

Fig. 4. The code of the $\mathcal{F}'_{\text{SMT}}$ functionality (adapted from [6] to mimic TLS messages).

On Using rid Instead of uid: It is important that the server gets a unique identifier with which they can link account information such as the user's bank account. We chose to replace the environment supplied uid by a rid that is randomly chosen by $\mathcal{F}'_{2\text{FA}}^{\text{out}}$ in the enrolment phase. This reduces the options of \mathcal{Z} , as now \mathcal{Z} cannot make two honest clients enrol with the same identifier. Of course the server can still store the user's email address alongside to the user record.

On (Not) Including an Interface for the Adversary to Guess the Biometric: Other authentication functionalities such as in [11, 16] have an interface that \mathcal{A} can use to make online or offline password guesses. Our functionality does not have an interfaces for offline guesses of credentials, because even if the server is corrupted, our protocol protects against such an attack. This is possible, because as opposed to [11, 16] in our case the client stores secret key material. Looking ahead, the only way that \mathcal{A} could perform an offline attack is after both corrupting the server and the client *and* breaking the security of the client's secure hardware module. Note that it is not enough to corrupt the client and break the security of the hardware module, because all data, which the hardware module stores, are independent of the biometrics. Regardless, our protocol builds on the assumption that the secure hardware is indeed secure and, thus, such an attack is out of scope of our model.

The TRYIMPERSONATE-interface can be seen as a way of allowing \mathcal{A} to perform online credential guesses for specific clients. Note, however, that this is only possible after \mathcal{A} has corrupted the respective client.

On Using Passwords Instead of Biometrics: Although our ideal functionality in Fig. 2 is aimed at capturing biometric authentication, it does in fact also capture password based authentication. Whether the inputs to the interfaces for the functionality are biometric templates or passwords does not matter. In order to perform equality checks on the passwords, one simply has to define the $\text{out}(\cdot, \cdot)$ function to return 1 if both arguments are equal (i.e. the passwords match), and 0 otherwise. Additionally our functionality naturally captures typo tolerance in the password matching. For that, the $\text{out}(\cdot, \cdot)$ function would be set to return 1 if the passwords match except for some common typos. This increases the applicability of our functionality to a wider range of scenarios.

Discussion of the Security Guarantees: The functionality $\mathcal{F}_{2\text{FA}}^{\text{out}}$ guarantees that an attacker cannot impersonate an honest client without corrupting that client *and* having a matching biometric template. The only way for \mathcal{A} to impersonate a client is through the TRYIMPERSONATE interface, after calling the CORRUPT interface.

The functionality also guarantees that nobody learns anything about the biometric templates, except of what can be inferred from the out function. This can be seen by observing that $\mathcal{F}_{2\text{FA}}^{\text{out}}$'s behavior does not depend on the biometric templates and the only way a biometric template appears in $\mathcal{F}_{2\text{FA}}^{\text{out}}$'s output is within $\text{out}(\mathbf{b}, \mathbf{b}')$. This holds even if the server is malicious and an arbitrary number of clients is corrupted. The same holds for the impossibility of offline guessing attacks, as $\mathcal{F}_{2\text{FA}}^{\text{out}}$ simply does not allow them. However, if the adversary corrupts both a client *and* the server, then they can try to impersonate that client with arbitrary biometric templates and at the same time learn the result of the out -function. When the out -function is the distance of the two templates, then the adversary can often reconstruct the corrupted client's reference template from these results. Note however, that this is only possible when both the client and the server are corrupted. Under these circumstances many protocols do not retain any security guarantees. Our protocol in Sect. 4.3 achieves these strong guarantees by using secure hardware on the client side.

Although the ideal functionality is for a single server, the composition theorem of [6] guarantees security also in situations with multiple servers, when there is one instance of $\mathcal{F}_{2\text{FA}}^{\text{out}}$ per server. A single user can also have multiple accounts at possibly different servers. In that case the user's device would run multiple instances of the protocol in Fig. 6. Again the composition theorem of [6] ensures that the security guarantees of $\mathcal{F}_{2\text{FA}}^{\text{out}}$ still hold.

4 The Framework

In this section we describe our general protocol $\Pi_{2\text{FA}}$ and show that it UC realizes $\mathcal{F}_{2\text{FA}}^{\text{out}}$. We start by describing a concrete use case in which our protocol could be used in practice. We then define several algorithms that are used in our protocol and the security proof and define their correctness. Every instantiation of our general protocol needs to instantiate these algorithms. Then we describe our general protocol and prove its security.

4.1 Use Case

In our use case in Fig. 5 the user has a device such as a smartphone or laptop (the host in Fig. 6) and wants to authenticate to a server. The secret keys of the FE scheme and the signature scheme are stored in secure hardware. This can be an external hardware token as in Fig. 5, or a trusted execution environment on the phone such as ARM’s TrustZone. For enrolment and authentication the device takes the user’s biometric template and gives it to the secure hardware. The secure hardware can optionally do liveness detection, i.e. trying to detect if the biometric data is coming from a live person or if e.g. somebody is holding a photo in front of the camera. In the next step, the secure hardware encodes and encrypts the biometric template and sends the resulting message to the server. The server then performs the enrolment or authentication. Thus, the client can authenticate to the server in a secure and privacy preserving manner. The first factor of the authentication is the possession of the hardware token and the second factor is the user’s biometric. When an attacker compromises the user’s host device or steals the hardware token, they cannot impersonate the user, as they are lacking the user’s biometric.

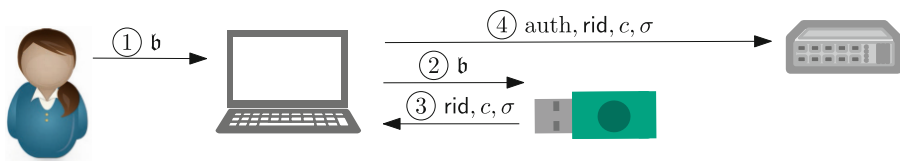


Fig. 5. The authentication phase of our use case where the secret keys are stored on a secure hardware token.

This use case is very similar to the setting of FIDO2 [12]. However, in FIDO2 the biometric matching is performed in the secure hardware. Both with our protocol in Sect. 4.3 and with FIDO2, if the attacker manages to compromise the keys in the secure hardware, then they can impersonate the user. However, an advantage of our protocol over FIDO2 is that in this case the user’s biometric reference template remains secret, because the data stored on the secure hardware do not depend on the reference template.

4.2 Requirements

Let $Sig = (Sig.Gen, Sig.Sign, Sig.Vfy)$ be a EUF-CMA secure signature scheme. Let $FE = (FE.Setup, FE.KeyGen, FE.Enc, FE.Dec)$ be a fh-IND secure FE scheme for the family of functions \mathcal{F} , the input space \mathcal{X} and output space \mathcal{Y} . To instantiate our framework the following six algorithms have to be defined and fulfill certain properties:

- $encodeRef(\cdot): \mathfrak{B} \rightarrow \mathcal{F}$, takes the reference template and encodes it as function of the FE scheme

- $\text{encodeProbe}(\cdot): \mathfrak{B} \rightarrow \mathcal{X}$, takes the probe template and encodes it as input of the FE scheme
- $\text{out}(\cdot, \cdot): \mathfrak{B} \times \mathfrak{B} \rightarrow \mathcal{D}$, takes two biometric templates and outputs the authentication result
- $\text{FE2out}(\cdot): \mathcal{Y} \rightarrow \mathcal{D}$, takes an output of FE.Dec and converts it to an authentication result
- $\text{chooseFakeRef}()$, outputs a fake reference template to be used by the simulator
- $\text{chooseFakeProbe}(\cdot, \cdot): \mathfrak{B} \times \mathcal{D} \rightarrow \mathfrak{B}$, takes a reference template and a desired result and outputs a corresponding fake probe template

We say that $(\text{encodeRef}, \text{encodeProbe}, \text{out}, \text{FE2out})$ are correct if $\forall \mathbf{b}, \mathbf{b}' \in \mathfrak{B}, \text{msk} \leftarrow \text{FE.Setup}(1^\lambda), \text{sk}_{\mathbf{b}} \leftarrow \text{FE.KeyGen}(\text{msk}, \text{encodeRef}(\mathbf{b})), c \leftarrow \text{FE.Enc}(\text{msk}, \text{encodeProbe}(\mathbf{b}'))$:

$$\text{out}(\mathbf{b}, \mathbf{b}') = \text{FE2out}(\text{FE.Dec}(\text{sk}_{\mathbf{b}}, c)).$$

We say that $(\text{chooseFakeRef}, \text{chooseFakeProbe})$ are correct if $\forall d \in \mathcal{D}, \mathbf{b} \leftarrow \text{chooseFakeRef}(), \mathbf{b}' \leftarrow \text{chooseFakeProbe}(\mathbf{b}, d): d = \text{out}(\mathbf{b}, \mathbf{b}')$. Choosing fake templates will later be necessary for the simulator and allows us to rely on the weaker fh-IND security instead of simulation based security of the FE scheme. We will instantiate these algorithms in Sect. 5.

4.3 The Protocol

The code of the client and the server of our protocol $\Pi_{2\text{FA}}$ are depicted in Fig. 6 and Fig. 7. We divide the client in two parts, the *host* and the *secure hardware*. The host is the normal program on the laptop or the phone, whereas the code of the secure hardware runs in a trusted execution environment or on an external hardware token. We use $\mathcal{F}'_{\text{SMT}}$ to model TLS messages sent by the client to the server as described in Fig. 4 and Sect. 3.2. Note that in our version of $\mathcal{F}'_{\text{SMT}}$ (as opposed to \mathcal{F}_{SMT} from [6]), the client's identity \mathcal{C} is *not* given to the server.

For enrolling, the host simply forwards the instruction to the secure hardware, which chooses a random rid and keys for the signature scheme and FE scheme. The secure hardware then encodes the biometric reference template and generates a FE key for the encoded template. It gives the rid , the FE key and the signature public key to the host, which sends it to the server. The server simply stores these data. The secure hardware stores the FE master secret key, the signature secret key and rid .

For authenticating, the host again passes the instruction to the secure hardware, which first checks if it is already enrolled. If so, it encodes the probe template and encrypts it with the FE scheme. Furthermore it signs $(\text{sid}, \text{ssid}, \text{rid}, c)$ in order to prove ownership of the signature secret key. The secure hardware gives the rid , the FE ciphertext and the signature to the host, who sends the message to the server. The server checks the signature and computes the output of the protocol. We assume that ssid is unique per client i.e. per rid . Thus, the value $(\text{sid}, \text{ssid}, \text{rid})$ is globally unique and prevents an attacker from reusing the signature. In practice the sid can be set as the server's name and the ssid as a counter

that is increased by the client at each authentication. The server would also keep a counter per client and only accept messages with a counter value higher than the stored one. Upon receiving the signature and the encrypted probe template, the server checks the signature and uses the FE.Dec algorithm to compute the output, which for example could be the distance of the biometric templates or a yes-no decision.

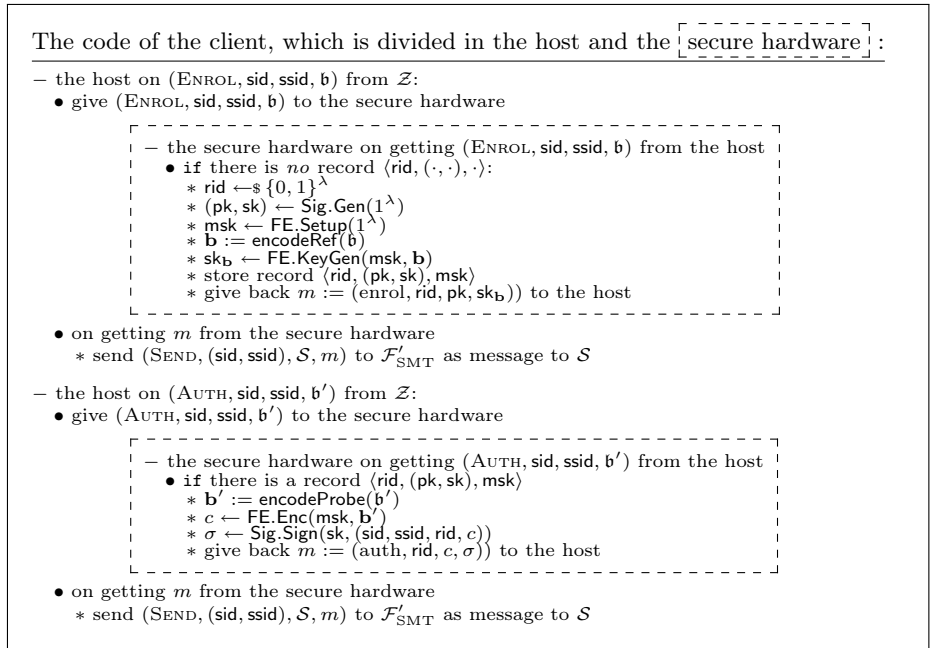


Fig. 6. The code of the client.

On the Need for Secure Hardware: Without secure hardware, the adversary can —by compromising the client— get the secret keys and they could be able to choose a fake encoding of a biometric template, which may convince the server that the attacker is authentic. For example when we instantiate the FE scheme with an IPFE scheme, as we do in Sect. 5, then the attacker can choose $\mathbf{b}' = 0 \dots 0^\top$. This makes the inner-product with the encoded reference template to be 0, which may convince the server that the distance is zero and allows the attacker to impersonate the user. This problem seems to be inherent to IPFE schemes and is also present in e.g. [7, 18]. We chose to address this problem by storing the client’s secret keys in secure hardware and letting the secure hardware do the encoding of the biometric template. This ensures that only correctly encoded templates are encrypted and signed. Another approach to prevent this attack is to add zero knowledge proofs to the protocol. The client

The code of the server:

- on (SENT, (sid, ssid), $m = (\text{enrol}, \text{rid}, \text{pk}, \text{sk}_b)$) from $\mathcal{F}'_{\text{SMT}}$ as message from some client:
 - if there is no record $\langle \text{rid}, \cdot, \cdot \rangle$:
 - * store $\langle \text{rid}, \text{pk}, \text{sk}_b \rangle$
 - * output (ENROL, sid, ssid, rid)
 - else output (ENROL, sid, ssid, \perp)
- on (SENT, (sid, ssid), $m = (\text{auth}, \text{rid}, c, \sigma)$) from $\mathcal{F}'_{\text{SMT}}$ as message from some client:
 - if there is a record $\langle \text{rid}, \text{pk}, \text{sk}_b \rangle$ and $\text{Sig.Vfy}(\text{pk}, (\text{sid}, \text{ssid}, \text{rid}, c), \sigma) = 1$:
 - * $d := \text{FE2out}(\text{FE.Dec}(\text{sk}_b, c))$
 - * output (AUTH, sid, ssid, rid, d)
 - else output (AUTH-FAIL, sid, ssid)

Fig. 7. The code of the server.

would then have to prove to the server that the biometric template has been correctly encoded before being encrypted.

4.4 Security Proof

Theorem 1. *If FE is a (single-key) fh-IND secure fh-FE scheme and Sig is an EUF-CMA secure signature scheme, then Π_{2FA} UC-emulates $\mathcal{F}_{2FA}^{\text{out}}$ in the $\mathcal{F}'_{\text{SMT}}$ hybrid model in the presence malicious adversaries.*

We defer the security proof to Appendix A.

5 Instantiation

Function hiding IPFE schemes allow computing inner-products between two encrypted vectors. This enables us to evaluate any distance measure between biometric templates that can be written as inner-product. When we then use fh-IPFE schemes in our framework, this enables the server to compute the distance between the client’s reference and probe template. In this section we describe how to use an fh-IPFE scheme IPFE to instantiate our framework for computing the Euclidean distance, the Hamming distance or the cosine similarity of two biometric templates. We also sketch how the techniques of [5] can be used to instantiate our framework in a way that avoids the leakage of the distance of the biometric templates.

Our framework can be instantiated with both [7, 18]. Both provide function hiding simulation security, which implies fh-IND security which is the notion that we need in our framework. The IPFE scheme of Cheon et al. [7] only allows the creation of a single decryption key, which is sufficient in our setting and allows the scheme to be quite efficient. The scheme relies on the security of the learning with errors assumption. The IPFE scheme of Kim et al. [18] allows an arbitrary number of decryption keys, which makes it less efficient. It uses pairings and has a security proof in the generic group model. We use the IPFE scheme as black-box, the parties call the IPFE.Setup, IPFE.KeyGen, IPFE.Enc,

IPFE.Dec algorithms whenever the respective algorithm in Fig. 6 and Fig. 7 is called. We will assume that our biometric templates \mathbf{b} are already embedded in e.g. Euclidean or Hamming space, i.e. they are vectors where a small distance implies similar biometrics. This is often achieved by applying a neural network to the biometric reading [9, 23]. Letting \mathcal{Z} directly choose the embedding can be seen as modeling \mathcal{Z} 's influence on the network creation and training. Next we show how to instantiate the algorithms described in Sect. 4.2.

Squared Euclidean Distance: In [23] the authors show how to transform face biometrics into vectors, where similar faces have low squared Euclidean distance. Let the discretized embedding be vectors in \mathbb{Z}_{m+1}^n (in [23] $n = 128$). Then the maximum squared Euclidean distance is $m^2 \cdot n$ and we can set $\mathcal{D} := \{0, \dots, m^2 \cdot n\}$. To later simplify the chooseFakeRef and chooseFakeProbe algorithms of the simulator we set $\mathfrak{B} := \mathbb{Z}_q^n$, for $q > m^2 \cdot n$. Note that this also allows \mathcal{Z} to choose biometric templates with too large coordinates, however, this does not help in finding a vector that is close to the reference template and, thus, does not impact security. Because the desired output is the squared Euclidean distance d_E , we define $\text{out}(\mathbf{b}, \mathbf{b}') := \min(d_E(\mathbf{b}, \mathbf{b}'), m^2 \cdot n)$. Note that the squared Euclidean distance between \mathbf{b} and \mathbf{b}' can also be written as $d_E(\mathbf{b}, \mathbf{b}') = -2\langle \mathbf{b}, \mathbf{b}' \rangle + \|\mathbf{b}\|^2 + \|\mathbf{b}'\|^2$. Thus, we can define $\text{encodeRef}(\mathbf{b}) := \mathbf{b} = \mathbf{b}_1 \dots \mathbf{b}_n \ 1 \ \|\mathbf{b}\|^2$ and $\text{encodeProbe}(\mathbf{b}') := \mathbf{b}' = -2\mathbf{b}'_1 \dots -2\mathbf{b}'_n \ \|\mathbf{b}'\|^2 \ 1$. Then $\langle \mathbf{b}, \mathbf{b}' \rangle$ is the squared Euclidean distance of \mathbf{b} and \mathbf{b}' . This requires an IPFE scheme with $\mathcal{X} = \mathbb{Z}_q^{n+2}$ and $\mathcal{Y} = \mathbb{Z}_q$. Finally we define $\text{FE2out}(d) := \min(d, m^2 \cdot n)$. Capping the result at $m^2 \cdot n$ is necessary to ensure that it stays inside of \mathcal{D} . Kim et al. [18] used the same encoding technique for computing similarity of text documents.

Next we explain how to instantiate chooseFakeRef and chooseFakeProbe. Let $\text{chooseFakeRef}() := \mathbf{b} := 0 \dots 0 \in \mathbb{Z}_q^n$. Then, finding a fake probe template with distance d , is the same as finding a vector $\mathbf{b}'_1 \dots \mathbf{b}'_n$ s.t. $\sum_{i \in [n]} \mathbf{b}'_i{}^2 = d$. When n is at least 4, then the existence of such a vector is guaranteed by Lagrange's four square theorem, furthermore there exist efficient algorithms for computing these four values [22]. We define chooseFakeProbe as setting the first four coordinates of \mathbf{b}' as the output of the algorithm of [22] and setting the other coordinates to 0. Note that chooseFakeRef and chooseFakeProbe are only part of the simulator and never need to be actually implemented or executed.

Hamming Distance: Iris readings can be transformed into binary vectors where a small Hamming distance corresponds to similar irises, e.g. with techniques described in [5]. For computing the Hamming distance d_H we use the same encoding technique as Kim et al. [18]. We assume that our templates are binary vectors $\mathbf{b} \in \mathfrak{B} := \{0, 1\}^n$. Then the largest possible Hamming distance is n , so we define $\mathcal{D} := \{0, \dots, n\}$. We require from the IPFE scheme that $\mathcal{X} = \mathbb{Z}_q^n$ and $\mathcal{Y} = \mathbb{Z}_q$, for $q > n$. Furthermore, we define $\text{out}(\mathbf{b}, \mathbf{b}') := d_H(\mathbf{b}, \mathbf{b}')$. For encoding a template \mathbf{b} we define the vector \mathbf{b} via $b_i = 1$, if $\mathbf{b}_i = 1$ and $b_i = -1$ if $\mathbf{b}_i = 0$. For example $1, 0, 1$ becomes $1, -1, 1$. We can then define $\text{encodeRef}(\mathbf{b}) := \text{encodeProbe}(\mathbf{b}) := \mathbf{b}$. However, the inner-product $\langle \mathbf{b}, \mathbf{b}' \rangle$ is not

equal to the Hamming distance $d_H(\mathbf{b}, \mathbf{b}')$. Let ip be the inner-product output by the IPFE.Dec algorithm. Then we define $\text{FE2out}(\text{ip}) := \max(\frac{n-\text{ip}}{2}, 0)$, to convert the inner-product into the Hamming distance.

Defining the algorithms for the simulator is relatively simple. We define $\text{chooseFakeRef}() := \mathbf{b} := 1 \dots 1^\top \in \mathfrak{B}$ and $\text{chooseFakeProbe}(\mathbf{b}, d)$ as the same vector with d of the entries flipped to 0.

Cosine Similarity: As noted in [2], the cosine similarity can also be used for biometric matching (e.g. for faces as in [20]) and can be easily computed as an inner-product. The cosine similarity of two vectors \mathbf{v}, \mathbf{w} is defined as $d_C(\mathbf{v}, \mathbf{w}) := \frac{\langle \mathbf{v}, \mathbf{w} \rangle}{\|\mathbf{v}\| \|\mathbf{w}\|}$, whereby the largest possible cosine similarity is 1, so we define $\mathcal{D} := [0, 1] \subset \mathbb{R}$. Close vectors have a high cosine similarity. We assume that the templates are vectors in $\mathfrak{B} := \mathbb{Z}_m^n$, with a fixed public L2 norm l . We require $\mathcal{X} = \mathbb{Z}_q^n, \mathcal{Y} = \mathbb{Z}_q$, for $q > l^2$. We define $\text{out}(\mathbf{b}, \mathbf{b}') := d_C(\mathbf{b}, \mathbf{b}')$ and $\text{encodeRef}(\mathbf{b}) := \text{encodeProbe}(\mathbf{b}) := \mathbf{b}$. Then the cosine similarity can be computed as $d_C(\mathbf{b}, \mathbf{b}') = \text{FE2out}(\langle \mathbf{b}, \mathbf{b}' \rangle) := \min(\frac{\langle \mathbf{b}, \mathbf{b}' \rangle}{l^2}, 1)$.

Defining chooseFakeRef and chooseFakeProbe such that the templates have the correct distance, satisfy the norm bound l and still have integer coordinates seems rather complex and we leave it as future work. Thus, our security proof does not cover the cosine similarity instantiation.

Checking if the Distance is Below a Threshold: The above constructions leak the distance of the biometric templates to the server, which can be avoided by using a construction of [5]. They use orthogonality functional encryption to check if the inner-product is equal to a certain value and further use it to check if the distance of two encrypted biometric templates is within a certain range i.e. below a threshold. Their technique works with both the Euclidean and the Hamming distance and can be used to instantiate our framework so that the server only learns the yes-no authentication result.

6 Implementation

In this section we will briefly describe our proof of concept implementation and the results of our performance tests. The source code is available at: <https://github.com/johanernst/ipfe-bio-auth>. This includes the Golang source code of the protocol and the performance tests, the Python code for creating the plots and the raw output data of the experiments. For the function hiding IPFE scheme we used the implementation of the scheme of [18] in the GOFE library described in [21]. We run the performance tests on a single thread of an Intel Core i5-10210U CPU on a laptop. As biometric templates we used random vectors with increasing number of elements. Each element is an integer value between 0 and 255 and we used the instantiation for the squared Euclidean distance described in Sect. 5. The choice of parameters is motivated by [23] who constructed a neural network that embeds face images into Euclidean space and has a very

high accuracy. They use float vectors with 128 entries and report that “...it can be quantized to 128-bytes without loss of accuracy.”

For each of the different template lengths we performed 3 runs, where each run consisted of enrolling/authenticating 10 clients. Finally we took the average to get the running time of a single call to each of the algorithms. We also separately measured the time spent in the calls to the IPFE schemes with the same number of runs and clients. The results are depicted in Fig. 8 and Table 1.

The experiments show that most algorithms are rather fast (below 0.25 ms), only the enrolment algorithm of the client is a bit slower (about 1.4s), because the Setup algorithm of the underlying IPFE scheme [18] requires the inversion of a $n \times n$ matrix, where n is the length of the vector. However, the enrolment is only performed once and the 1.4s (for templates with 128 entries as in [23]) are unlikely to significantly disturb the user experience. The time that the server needs to enrol a client is so low because the server does not have to run any IPFE operations. Furthermore, Table 1 shows that most of the running time is consumed by the IPFE scheme. This means that our protocol profits significantly from future improvements in the area of function hiding IPFE. Also our scheme only needs *single-key* function hiding IPFE which can probably be constructed much more efficiently. Therefore, instantiating our protocol with a *single-key* function hiding IPFE will likely also boost efficiency. For templates with 128 entries the size of the enrolment message is 16.47 KB and the size of the authentication message is 33.29 KB.

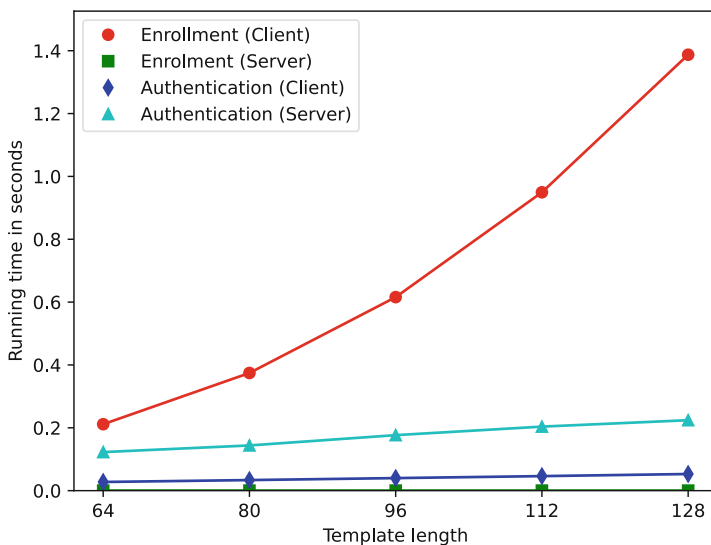


Fig. 8. The running time of the different parts of our protocol with different template lengths.

Table 1. This table shows the exact running times in seconds of the four algorithms of our protocol. In parenthesis there is the time spent in the calls to the IPFE scheme.

Template size	Client Enrolment	Server Enrolment	Client Authentication	Server Authentication
64	0.211 (0.210)	0.0001 (-)	0.028 (0.027)	0.123 (0.121)
128	1.387 (1.389)	0.0001 (-)	0.053 (0.054)	0.224 (0.224)

7 Conclusion and Future Work

Privacy-preserving biometric authentication is a complex problem and although significant work has been proposed in the area, the respective security models often do not capture all desired security goals or do not model all realistic adversarial behavior. In this paper we discussed and highlighted the importance of both *privacy preserving* and *secure* biometric authentication systems that maintain their security guarantees in complex real world settings. To this end we proposed an ideal functionality for universally composable biometric based two-factor authentication. To the best of our knowledge this is the first description of an ideal functionality for biometric based two-factor authentication in the UC framework. Furthermore, we proposed a general protocol for privacy preserving biometric authentication that provides UC security guarantees and can be instantiated using any suitable function hiding functional encryption scheme and a signature scheme. We provide a detailed security analysis and proof of the proposed general framework. Additionally, we showed how to concretely instantiate our framework with a function hiding IPFE scheme and, thereby, allow the computation of the Euclidean distance, the Hamming distance or the cosine similarity. Finally we explained our proof of concept implementation and presented the results of the performance tests.

Future Work: A worthwhile direction for future work may be to use our ideal functionality to analyze the FIDO2 protocol in the UC framework. The functionality would need to be adapted a little bit. For example it would probably need an interface for the server to indicate that they are willing to participate in the protocol and an interface for the adversary to allow the delivery of the server’s message to the client.

Another interesting direction would be to extend our model to allow corrupted clients to get uncorrupted again. This can happen when the malware is removed from the client’s device, or when the client gets back their, previously stolen, hardware token.

Acknowledgements. This work was partially funded by the EU-funded Marie Curie ITN TReSPAS-ETN project under the grant agreement 860813. We would like to thank the anonymous reviewers of ACNS for their detailed and helpful comments and suggestions and Astrid Ottenhues for helpful discussions.

A Security Proof

Below we give the proof of Theorem 1.

Proof. We first provide a simulator in Fig. 9 and Fig. 10. Then we show that no PPT environment \mathcal{Z} can distinguish between the real world, where it is interacting with the honest parties and the dummy adversary, from the ideal world, where it is interacting with the honest parties and the simulator. We do so by considering all actions that \mathcal{Z} can take and argue for each of them that the results, which \mathcal{Z} gets in the real world and the ideal world, are essentially the same. The actions that \mathcal{Z} can take are:

- (ENROL, sid, ssid, \mathbf{b}) to an honest client
- (AUTH, sid, ssid, \mathbf{b}') to an honest client
- (OK, (sid, ssid)) to $\mathcal{F}'_{\text{SMT}}$
- (SEND, (sid, ssid), \mathcal{S} , $m = (\text{enrol}, \text{rid}, \text{pk}, \text{sk}_{\mathbf{b}})$) to $\mathcal{F}'_{\text{SMT}}$ in the name of a corrupted client
- (SEND, (sid, ssid), \mathcal{S} , $m = (\text{auth}, \text{rid}, c, \sigma)$) to $\mathcal{F}'_{\text{SMT}}$ in the name of a corrupted client
- (CORRUPT, sid) to a client \mathcal{C}
- (TRYIMPERSONATE, sid, ssid, \mathbf{b}') to a client \mathcal{C}

To simplify the presentation, we assume that \mathcal{Z} does not delay or block messages, whenever the sender or receiver is corrupted. In that case the receiver directly gets the message without the need for \mathcal{Z} to send (OK, (sid, ssid)) to $\mathcal{F}'_{\text{SMT}}$. This is reasonable, because \mathcal{Z} cannot gain anything from blocking its own messages. Therefore, when \mathcal{S} is corrupted, Sim can directly send (ENROLOK, sid, ssid) (resp. (AUTHOK, sid, ssid)) to $\mathcal{F}^{\text{out}}_{2\text{FA}}$ after receiving (ENROL, sid, ssid, \cdot) (resp. (AUTH, sid, ssid, \cdot)) from $\mathcal{F}^{\text{out}}_{2\text{FA}}$. In the real world we say that rid is *enroled*, if the server has a record $\langle \text{rid}, \cdot, \cdot \rangle$. In the ideal world we say that rid is *enroled*, if the ideal functionality has a record $\langle \text{enroled}, \cdot, \text{rid}, \cdot \rangle$ or $\langle \text{enroled-adversarial}, \text{rid}, \cdot \rangle$. In both worlds this is equivalent to the server having output (ENROL, sid, ssid, rid), for some sid and ssid.

The simulator uses five different tables. Table T_1 is for pending messages, T_2 contains entries for the adversarially enroled clients. In case the server is corrupted, T_3 contains an entry for each of the enroled clients. Table T_4 contains an entry for each client that was adaptively corrupted by \mathcal{Z} and T_5 contains all (fake) messages that Sim created as response to TRYIMPERSONATE instructions from \mathcal{Z} .

- (ENROL, sid, ssid, \mathbf{b}) to an honest client \mathcal{C} : \mathcal{Z} calls the enrol-interface of \mathcal{C} .

Case 1. The server is honest:

```

1 : Let  $l_e, l_a$  be the length of the enrolment- and authentication messages respectively.
2 : Forwarding messages:
3 :   - on (ENROL, sid, ssid,  $\mathcal{C}$ ) from  $\mathcal{F}_{2FA}^{out}$  :
4 :     • if  $\mathcal{S}$  is corrupted: send (ENROLOK, sid, ssid) to  $\mathcal{F}_{2FA}^{out}$ 
5 :     • else:
6 :       * simulate  $\mathcal{F}'_{SMT}$  by sending (SENT, (sid, ssid),  $\mathcal{C}, \mathcal{S}, l_e$ ) to  $\mathcal{Z}$  as message from  $\mathcal{F}'_{SMT}$ 
7 :       * store (enrol, ssid) in table  $T_1$ 
8 :   - on (AUTH, sid, ssid,  $\mathcal{C}$ ) from  $\mathcal{F}_{2FA}^{out}$  :
9 :     • if  $\mathcal{S}$  is corrupted: send (AUTHOK, sid, ssid) to  $\mathcal{F}_{2FA}^{out}$ 
10 :    • else:
11 :      * simulate  $\mathcal{F}'_{SMT}$  by sending (SENT, (sid, ssid),  $\mathcal{C}, \mathcal{S}, l_a$ ) to  $\mathcal{Z}$  as message from  $\mathcal{F}'_{SMT}$ 
12 :      * store (auth, ssid) in table  $T_1$ 
13 :   - on (OK, (sid, ssid)) from  $\mathcal{Z}$  to  $\mathcal{F}'_{SMT}$ :
14 :     • if there is a record (enrol, ssid) in table  $T_1$ : send (ENROLOK, sid, ssid) to  $\mathcal{F}_{2FA}^{out}$ 
15 :     • else if there is a record (auth, ssid) in table  $T_1$ : send (AUTHOK, sid, ssid) to  $\mathcal{F}_{2FA}^{out}$ 
16 :     • else: ignore this message
17 : Client messages to  $\mathcal{F}'_{SMT}$ :
18 :   - on (SEND, (sid, ssid),  $\mathcal{S}, m = (\text{enrol}, \text{rid}, \text{pk}, \text{sk}_b)$ ) from  $\mathcal{Z}$  to  $\mathcal{F}'_{SMT}$  (from corrupted client)
19 :     • if server  $\mathcal{S}$  is corrupted: send (SENT, (sid, ssid),  $m$ ) to  $\mathcal{Z}$  as  $\mathcal{F}'_{SMT}$ 's output to  $\mathcal{S}$ 
20 :     • else:
21 :       * choose  $b \leftarrow \text{chooseFakeRef}()$ 
22 :       * send (ENROL, sid, ssid, rid,  $b$ ) to  $\mathcal{F}_{2FA}^{out}$  //using the adversary's interface
23 :       * if record (rid,  $\cdot, \cdot, \cdot$ ) does not exist in table  $T_2$ : store (rid,  $b$ , pk,  $\text{sk}_b$ ) in table  $T_2$ 
24 :   - on (SEND, (sid, ssid),  $\mathcal{S}, m = (\text{auth}, \text{rid}, c, \sigma)$ ) from  $\mathcal{Z}$  to  $\mathcal{F}'_{SMT}$  (from corrupted client)
25 :     • if server  $\mathcal{S}$  is corrupted: send (SENT, (sid, ssid),  $m$ ) to  $\mathcal{Z}$  as  $\mathcal{F}'_{SMT}$ 's output to  $\mathcal{S}$ 
26 :     • else:
27 :       * if there is entry (rid,  $b$ , pk,  $\text{sk}_b$ ) in table  $T_2$  and  $\text{Sig.Vfy}(\text{pk}, \sigma, (\text{sid}, \text{ssid}, \text{rid}, c)) = 1$ :
28 :         .  $d := \text{FE2out}(\text{FE.Dec}(\text{sk}_b, c))$ 
29 :         . choose  $b' \leftarrow \text{chooseFakeProbe}(b, d)$  //make sure that  $\text{out}(b, b') = d$ 
30 :         . send (AUTH, sid, ssid, rid,  $b'$ ) to  $\mathcal{F}_{2FA}^{out}$  //using the adversary's interface
31 :       * else if there is an entry (rid, sid, ssid,  $c$ , pk,  $\mathcal{C}, b'$ ) in  $T_5$ 
32 :         and  $\text{Sig.Vfy}(\text{pk}, \sigma, (\text{sid}, \text{ssid}, \text{rid}, c)) = 1$ :
33 :         . send (TRYIMPERSONATE, sid, ssid,  $\mathcal{C}, b'$ ) to  $\mathcal{F}_{2FA}^{out}$ 
34 :       * else: send (AUTH, sid, ssid,  $\perp, \perp$ ) to  $\mathcal{F}_{2FA}^{out}$  //using the adversary's interface
35 : Simulating a corrupted server:
36 :   - on (ENROL, sid, ssid, rid) from  $\mathcal{F}_{2FA}^{out}$  to the corrupted server:
37 :     •  $\text{msk} \leftarrow \text{FE.Setup}(1^\lambda)$ 
38 :     • choose  $b \leftarrow \text{chooseFakeRef}()$ 
39 :     •  $b := \text{encodeRef}(b)$ 
40 :     •  $\text{sk}_b \leftarrow \text{FE.KeyGen}(\text{msk}, b)$ 
41 :     •  $(\text{pk}, \text{sk}) \leftarrow \text{Sig.Gen}(1^\lambda)$ 
42 :     • store (rid,  $b$ , pk, sk, msk,  $\text{sk}_b$ ) in table  $T_3$ 
43 :     • send (SENT, (sid, ssid),  $m = (\text{enrol}, \text{rid}, \text{pk}, \text{sk}_b)$ ) to  $\mathcal{Z}$  as  $\mathcal{F}'_{SMT}$ 's output to  $\mathcal{S}$ 
44 :   - on (AUTH, sid, ssid, rid,  $d$ ) from  $\mathcal{F}_{2FA}^{out}$  to the corrupted server:
45 :     • retrieve record (rid,  $b$ , pk, sk, msk,  $\text{sk}_b$ ) from table  $T_3$ 
46 :     • choose  $b' \leftarrow \text{chooseFakeProbe}(b, d)$  //make sure that  $\text{out}(b, b') = d$ 
47 :     •  $b' := \text{encodeProbe}(b')$ 
48 :     •  $c \leftarrow \text{FE.Enc}(\text{msk}, b')$ 
49 :     •  $\sigma \leftarrow \text{Sig.Sign}(\text{sk}, (\text{sid}, \text{ssid}, \text{rid}, c))$ 
50 :     • send (SENT, (sid, ssid),  $m = (\text{auth}, \text{rid}, c, \sigma)$ ) to  $\mathcal{Z}$  as  $\mathcal{F}'_{SMT}$ 's output to  $\mathcal{S}$ 

```

Fig. 9. The code of the simulator.

```

51 : Continuation of the simulator's code
52 : - on (CORRUPT, sid) from  $\mathcal{Z}$  to  $\mathcal{C}$ :
53 :   • send (CORRUPT, sid,  $\mathcal{C}$ ) to  $\mathcal{F}_{2FA}^{out}$ 
54 :   • if  $\mathcal{F}_{2FA}^{out}$  answers with (CORRUPTED, sid, rid):
55 :     * if  $\mathcal{S}$  is corrupted:
56 :       · retrieve (rid, b, pk, sk, msk,  $sk_b$ ) from  $T_3$ 
57 :       · store ( $\mathcal{C}$ , rid, msk, pk, sk, b) in  $T_4$ .
58 :     * else: //if  $\mathcal{S}$  is not corrupted
59 :       · msk  $\leftarrow$  FE.Setup( $1^\lambda$ )
60 :       · (pk, sk)  $\leftarrow$  Sig.Gen( $1^\lambda$ )
61 :       · choose b  $\leftarrow$  chooseFakeRef()
62 :       · store ( $\mathcal{C}$ , rid, msk, pk, sk, b) in table  $T_4$ 
63 :     * send (CORRUPTED, sid) to  $\mathcal{Z}$ 

64 : - on (TRYIMPERSONATE, sid, ssid,  $b'$ ) to  $\mathcal{C}$ :
65 :   • retrieve record ( $\mathcal{C}$ , rid, msk, pk, sk, b) from  $T_4$ 
66 :   • if server  $\mathcal{S}$  is corrupted:
67 :     * send (TRYIMPERSONATE, sid, ssid,  $\mathcal{C}$ ,  $b'$ ) to  $\mathcal{F}_{2FA}^{out}$ 
68 :     * receive back (AUTH, sid, ssid, rid,  $d$ ) as output to  $\mathcal{S}$ 
69 :     *  $\hat{b}' \leftarrow$  chooseFakeProbe(b,  $d$ )
70 :     *  $\tilde{b}' :=$  encodeProbe( $\hat{b}'$ )
71 :     *  $c \leftarrow$  FE.Enc(msk,  $\tilde{b}'$ )
72 :     *  $\sigma \leftarrow$  Sig.Sign(sk, (sid, ssid, rid,  $c$ ))
73 :     * give (rid,  $c$ ,  $\sigma$ ) to  $\mathcal{Z}$  as the output of the secure hardware to the host
74 :   • else: //  $\mathcal{S}$  is not corrupted
75 :     *  $b' :=$  encodeProbe( $b'$ )
76 :     *  $c \leftarrow$  FE.Enc(msk,  $b'$ )
77 :     *  $\sigma \leftarrow$  Sig.Sign(sk, (sid, ssid, rid,  $c$ ))
78 :     * store (rid, sid, ssid,  $c$ , pk,  $\mathcal{C}$ ,  $b'$ ) in  $T_5$ 
79 :     * give (rid,  $c$ ,  $\sigma$ ) to  $\mathcal{Z}$  as the output of the secure hardware to the host

```

Fig. 10. The second part of the code of the simulator.

Real world: \mathcal{C} only continues if this is the first ENROL message they got. They execute the setup algorithm of the FE scheme and the signature scheme and choose a random rid. \mathcal{C} prepares the message m for the server and sends (SEND, (sid, ssid), \mathcal{S} , m) to \mathcal{F}'_{SMT} . \mathcal{F}'_{SMT} then sends (SENT, (sid, ssid), \mathcal{C} , \mathcal{S} , length(m)) to \mathcal{A} , who gives it to \mathcal{Z} .

Ideal world: \mathcal{C} sends (ENROL, sid, ssid, b) to \mathcal{F}_{2FA}^{out} , which only continues if this is the first ENROL message from \mathcal{C} . \mathcal{F}_{2FA}^{out} then sends (ENROL, sid, ssid, \mathcal{C} , \mathcal{S}) to Sim, who gives (SENT, (sid, ssid), \mathcal{C} , \mathcal{S} , l_e) to \mathcal{Z} .

In both worlds \mathcal{Z} gets a message if and only if the client has not yet sent an enrolment message. By definition of l_e , we have that length(m) = l_e and, therefore, the messages that \mathcal{Z} gets in both worlds are the same.

Case 2. The server is corrupted:

Real world: \mathcal{C} sends (SEND, (sid, ssid), \mathcal{S} , $m :=$ (enrol, rid, pk, sk_b)) to \mathcal{F}'_{SMT} for random rid and fresh pk and sk_b . \mathcal{S} directly gives this message to \mathcal{Z} .

Ideal world: \mathcal{F}_{2FA}^{out} sends (ENROL, sid, ssid, \mathcal{C}) to Sim, which replies with (ENROLOK, sid, ssid). Sim then gets (ENROL, sid, ssid, rid) as output to the corrupted server. Sim chooses b and generates pk and sk_b . They then give

$(\text{SENT}, (\text{sid}, \text{ssid}), m := (\text{enrol}, \text{rid}, \text{pk}, \text{sk}_{\mathbf{b}}))$ to \mathcal{Z} in the name of the corrupted server.

In both worlds sid and ssid are the same and rid is a random value that \mathcal{Z} has not previously seen. Also pk is in both worlds the result of Sig.Gen . The only critical part is $\text{sk}_{\mathbf{b}}$. In the real world the underlying vector \mathbf{b} is the user's biometric, whereas in the ideal world the simulator chose $\mathbf{b} \leftarrow \text{chooseFakeRef}()$. However, both $\text{sk}_{\mathbf{b}}$ are indistinguishable due to the function hiding property of the FE scheme. In Lemma 1 we give a reduction that breaks the fh-IND-security of FE if \mathcal{Z} can distinguish between the real and ideal world.

• **(AUTH, sid, ssid, \mathbf{b}') to an honest client \mathcal{C} :** \mathcal{Z} calls the auth-interface of \mathcal{C} .

Case 1. The server is honest:

Real world: \mathcal{C} checks if they are enrolled. If so, \mathcal{C} prepares the authentication message m for the server and sends $(\text{SEND}, (\text{sid}, \text{ssid}), \mathcal{S}, m)$ to $\mathcal{F}'_{\text{SMT}}$, which sends $(\text{SENT}, (\text{sid}, \text{ssid}), \mathcal{C}, \mathcal{S}, \text{length}(m))$ to \mathcal{A} , who forwards it to \mathcal{Z} .

Ideal world: \mathcal{C} sends $(\text{AUTH}, \text{sid}, \text{ssid}, \mathbf{b}')$ to $\mathcal{F}_{2\text{FA}}^{\text{out}}$. If \mathcal{C} has previously sent an enrol-message, $\mathcal{F}_{2\text{FA}}^{\text{out}}$ sends $(\text{AUTH}, \text{sid}, \text{ssid}, \mathcal{C}, \mathcal{S})$ to Sim, who gives $(\text{SENT}, (\text{sid}, \text{ssid}), \mathcal{C}, \mathcal{S}, l_a)$ to \mathcal{Z} .

In both worlds \mathcal{Z} gets a message if and only if the client has previously sent an enrolment message. By definition of l_a , we have that $\text{length}(m) = l_a$ and, therefore, the messages that \mathcal{Z} gets are the same in both worlds.

Case 2. The server is corrupted:

Real world: If \mathcal{C} has previously sent an enrol-message, they send $(\text{SEND}, (\text{sid}, \text{ssid}), \mathcal{S}, m = (\text{auth}, \text{rid}, c, \sigma))$ to $\mathcal{F}'_{\text{SMT}}$, where c is the encrypted, encoded \mathbf{b} and σ a signature of $(\text{sid}, \text{ssid}, \text{rid}, c)$. \mathcal{S} receives this message and directly gives it to \mathcal{Z} .

Ideal world: If \mathcal{C} has previously sent an enrol-message, $\mathcal{F}_{2\text{FA}}^{\text{out}}$ sends $(\text{AUTH}, \text{sid}, \text{ssid}, \mathcal{C})$ to Sim, which replies with $(\text{AUTHOK}, \text{sid}, \text{ssid})$. Sim then gets $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, d = \text{out}(\mathbf{b}, \mathbf{b}'))$ as output to the corrupted server, where \mathbf{b} and \mathbf{b}' are the client's reference and fresh template. Sim chooses a fake probe template such that its FE output with the fake reference template is exactly d . Sim then encodes and encrypts the new fake template and creates a signature, using the self-chosen keys from the enrolment phase, as an honest client would do. They then give $(\text{SENT}, (\text{sid}, \text{ssid}), m := (\text{auth}, \text{rid}, c, \sigma))$ to \mathcal{Z} in the name of the corrupted server.

In both worlds ssid is the same and rid is a random value that matches the rid from the enrolment phase. The critical component is the ciphertext c . Here we rely on the fh-IND-security of the FE scheme, which ensures that no PPT adversary can distinguish between two ciphertexts, if the output of $\text{FE.Dec}(\text{sk}_{\mathbf{b}}, \cdot)$ is the same in both cases. By choosing the fake templates as $\mathbf{b}' \leftarrow \text{chooseFakeProbe}(\mathbf{b}, d)$, Sim ensures that the FE outputs are the same in both worlds. We show the indistinguishability of both worlds formally in Lemma 1 by giving a reduction, which breaks the fh-IND-security of the FE scheme, if \mathcal{Z} is able to distinguish between the worlds. The signatures σ in both worlds are

indistinguishable, as the secret keys are identically distributed and the signed messages are indistinguishable.

- **(OK, (sid, ssid)) to $\mathcal{F}'_{\text{SMT}}$** : The environment lets through a client's message:

Case 1. (ssid belongs to an *enrol*-message of a client \mathcal{C}):

Real world: \mathcal{C} chose rid uniformly from $\{0,1\}^\lambda$, therefore, \mathcal{S} will not have a record $\langle \text{rid}, \text{pk}, \text{sk}_b \rangle$ with overwhelming probability. Thus, \mathcal{S} outputs $(\text{ENROL}, \text{sid}, \text{ssid}, \text{rid})$.

Ideal world: Sim sends $(\text{ENROLOK}, \text{sid}, \text{ssid})$ to $\mathcal{F}^{\text{out}}_{2\text{FA}}$. $\mathcal{F}^{\text{out}}_{2\text{FA}}$ will not have a record $\langle \text{enroled}, \mathcal{C}, \cdot, \cdot \rangle$, because \mathcal{C} has not yet enrolled and enrolls at most once. Hence, $\mathcal{F}^{\text{out}}_{2\text{FA}}$ will choose rid at random and give $(\text{ENROL}, \text{sid}, \text{ssid}, \text{rid})$ as output to \mathcal{Z} .

In both worlds rid is a random bit string that \mathcal{Z} has not seen before. Therefore, both worlds are indistinguishable for \mathcal{Z} .

Case 2. (ssid belongs to an *authentication*-message of a client \mathcal{C}): If \mathcal{Z} previously let through the corresponding enrol-message of \mathcal{C} then \mathcal{S} has a record $\langle \text{rid}, \text{pk}, \text{sk}_b \rangle$ and $\mathcal{F}^{\text{out}}_{2\text{FA}}$ has a record $\langle \text{enroled}, \mathcal{C}, \text{rid}, b \rangle$. Thus, in the real world \mathcal{Z} will get $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, d = \text{FE2out}(\text{FE.Dec}(\text{sk}_b, c)))$ from \mathcal{S} . In the ideal world \mathcal{Z} will get $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, \text{out}(b, b'))$, where b, b' are the same in both worlds (chosen by \mathcal{Z}). In both worlds rid will match the rid from the enrol-message. By correctness of $(\text{encodeRef}, \text{encodeProbe}, \text{out}, \text{FE2out})$ we have $d = \text{out}(b, b')$.

If \mathcal{Z} did not let through \mathcal{C} 's enrol-message, \mathcal{S} has no record $\langle \text{rid}, \cdot, \cdot \rangle$ (\mathcal{Z} does not even know rid) and $\mathcal{F}^{\text{out}}_{2\text{FA}}$ has no record $\langle \text{enroled}, \mathcal{C}, \cdot, \cdot \rangle$. Thus, in both worlds, \mathcal{Z} gets as output $(\text{AUTH-FAIL}, \text{sid}, \text{ssid})$.

- **(SEND, (sid, ssid), $m = (\text{enrol}, \text{rid}, \text{pk}, \text{sk}_b)$) to $\mathcal{F}'_{\text{SMT}}$** : A corrupted client's enrol-message:

Case 1. The server is honest:

Real world: If \mathcal{S} previously output $(\text{ENROL}, \text{sid}, \text{ssid}', \text{rid})$ (i.e. rid is already enrolled), then \mathcal{S} will output $(\text{ENROL}, \text{sid}, \text{ssid}, \perp)$. Otherwise \mathcal{S} will output $(\text{ENROL}, \text{sid}, \text{ssid}, \text{rid})$.

Ideal world: Sim uses the adversary-interface of $\mathcal{F}^{\text{out}}_{2\text{FA}}$ by sending $(\text{ENROL}, \text{sid}, \text{ssid}, \text{rid}, b)$, for a fake template b . If rid is already enrolled, $\mathcal{F}^{\text{out}}_{2\text{FA}}$ will output $(\text{ENROL}, \text{sid}, \text{ssid}, \perp)$ to \mathcal{S} and otherwise $(\text{ENROL}, \text{sid}, \text{ssid}, \text{rid})$. The outputs in both worlds are identical.

Case 2. The server is corrupted: In the real world \mathcal{S} will receive $(\text{SENT}, (\text{sid}, \text{ssid}), m)$ from $\mathcal{F}'_{\text{SMT}}$ and output it to \mathcal{Z} . In the ideal world Sim will give $(\text{SENT}, (\text{sid}, \text{ssid}), m)$ to \mathcal{Z} in the name of \mathcal{S} . In both worlds \mathcal{Z} gets identical output.

- **(SEND, (sid, ssid), $m = (\text{auth}, \text{rid}, c, \sigma)$) to $\mathcal{F}'_{\text{SMT}}$** : A corrupted client's auth-message:

Case 1. The server is honest: This is the case which shows that an attacker can first, not impersonate an honest client and second, still needs a valid biometric to impersonate an adaptively corrupted client.

First consider the case where rid is not enroled, or the signature σ is not valid. Then in both worlds \mathcal{Z} will get $(\text{AUTH-FAIL}, \text{sid}, \text{ssid})$ as output from \mathcal{S} .

Next, consider the case that the signature is valid *and* rid belongs to a client that has been enroled by \mathcal{A} , i.e. $\mathcal{F}_{2\text{FA}}^{\text{out}}$ has a record $(\text{enroled-adversarial}, \text{rid}, \cdot)$ and equivalently Sim has an entry $(\text{rid}, \cdot, \cdot, \cdot)$ in T_2 . In the real world, \mathcal{S} will output $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, d_{\text{real}})$ and in the ideal world \mathcal{S} will output $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, d_{\text{ideal}})$. We have $d_{\text{real}} = d_{\text{ideal}}$, because Sim computes its internal variable d exactly as the real server computes its output value and then uses it to generate fake templates that make $\mathcal{F}_{2\text{FA}}^{\text{out}}$ output $\text{out}(\mathbf{b}, \mathbf{b}')$ to \mathcal{S} . By correctness of $(\text{chooseFakeRef}, \text{chooseFakeProbe})$ Sim 's fake template $\mathbf{b}' \leftarrow \text{chooseFakeProbe}(\mathbf{b}, d)$ satisfies $\text{out}(\mathbf{b}, \mathbf{b}') = d$.

Now consider the case where rid is enroled, the signature is valid *and* Sim has an entry $(\text{rid}, \text{sid}, \text{ssid}, c, \cdot, \mathcal{C}, \mathbf{b}')$ in T_5 , where rid , sid , ssid and c are the same as from \mathcal{Z} 's message to $\mathcal{F}'_{\text{SMT}}$. This implies that \mathcal{Z} has corrupted \mathcal{C} and has sent a $(\text{TRYIMPERSONATE}, \text{sid}, \text{ssid}, \mathbf{b}')$ instruction to \mathcal{A}/Sim and is now instructing \mathcal{A}/Sim to send the message —that \mathcal{Z} got as response to the TRYIMPERSONATE instruction —to \mathcal{S} . Thus, in the real world \mathcal{S} will output $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, d_{\text{real}})$. In the ideal world Sim will send $(\text{TRYIMPERSONATE}, \text{sid}, \text{ssid}, \mathcal{C}, \mathbf{b}')$ to $\mathcal{F}_{2\text{FA}}^{\text{out}}$ which will then send $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, \text{out}(\mathbf{b}, \mathbf{b}'))$ to \mathcal{S} . Since in this case \mathbf{b} and \mathbf{b}' will be the same in both worlds, we have that $d_{\text{real}} = \text{out}(\mathbf{b}, \mathbf{b}')$, by correctness of $(\text{encodeRef}, \text{encodeProbe}, \text{out}, \text{FE2out})$.

Let us now consider the last case, where neither of the above is true, Sim gets to the else-case (in line 34) *and* rid is enroled *and* the signature is valid. In the ideal world, Sim will send $(\text{AUTH}, \text{sid}, \text{ssid}, \perp, \perp)$ to $\mathcal{F}_{2\text{FA}}^{\text{out}}$, which will then give $(\text{AUTH-FAIL}, \text{sid}, \text{ssid})$ as output to \mathcal{S} . In the real world, however, \mathcal{S} will output $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, d_{\text{real}})$. Thus, in this case \mathcal{Z} can distinguish between the worlds. However, this case can only occur if \mathcal{Z} forges a signature. In Lemma 2 we sketch a reduction that wins the EUF-CMA game in that case.

Case 2. The server is corrupted: Exactly as in the case of a corrupted client's enrol-message, in both worlds the server will output $(\text{SENT}, (\text{sid}, \text{ssid}), m)$.

• **Instruction to \mathcal{A}/Sim to send $(\text{CORRUPT}, \text{sid})$ to (the backdoor tape of) client \mathcal{C} :**

Real world: \mathcal{A} will send $(\text{CORRUPT}, \text{sid})$ to \mathcal{C} (on the backdoor tape). If and only if the client \mathcal{C} exists *and* is enroled, \mathcal{C} 's shell will answer with $(\text{CORRUPTED}, \text{sid})$. \mathcal{A} will then forward this message to \mathcal{Z} .

Ideal world: Sim will send $(\text{CORRUPT}, \text{sid}, \mathcal{C})$ to $\mathcal{F}_{2\text{FA}}^{\text{out}}$. $\mathcal{F}_{2\text{FA}}^{\text{out}}$ will answer with $(\text{CORRUPTED}, \text{sid}, \text{rid})$ if and only if the client \mathcal{C} exists *and* is enroled. In that case Sim will send $(\text{CORRUPTED}, \text{sid})$ to \mathcal{Z} .

Therefore, in both worlds \mathcal{Z} will get the output $(\text{CORRUPTED}, \text{sid})$ if and only if \mathcal{C} exists *and* is enroled. This is independent of whether the server is corrupted.

• **Instruction to \mathcal{A}/Sim to send $(\text{TRYIMPERSONATE}, \text{sid}, \text{ssid}, \mathbf{b}')$ to (the backdoor tape of) client \mathcal{C} :**

Case 1. The server is honest:

Real world: \mathcal{A} will send $(\text{TRYIMPERSONATE}, \text{sid}, \text{ssid}, \mathbf{b}')$ to \mathcal{C} (on the backdoor tape). If \mathcal{C} is corrupted, the shell will give $(\text{AUTH}, \text{sid}, \text{ssid}, \mathbf{b}')$ to the secure hardware, which will respond with $m = (\text{auth}, \text{rid}, c, \sigma)$. The shell will give m to \mathcal{A} who forwards it to \mathcal{Z} .

Ideal world: Sim will retrieve the fake keys from table T_4 which will exist only if \mathcal{C} has been corrupted. Then Sim will encode, encrypt and sign \mathbf{b}' as the secure hardware would have done and give $m = (\text{auth}, \text{rid}, c, \sigma)$ to \mathcal{Z} .

In both worlds \mathcal{Z} will get an answer if and only if \mathcal{C} has been corrupted before. In both worlds the rid is uniformly random, but stays the same over multiple TRYIMPERSONATE instructions. Furthermore, c and σ are generated with the same inputs and identically distributed keys which stay the same for multiple calls to TRYIMPERSONATE . Therefore, both worlds are perfectly indistinguishable.

Case 2. The server is corrupted:

Real world: \mathcal{Z} will get the same as in the case of an uncorrupted server, namely $m = (\text{auth}, \text{rid}, c, \sigma)$.

Ideal world: Sim will retrieve the fake keys from table T_4 which will exist only if \mathcal{C} has been corrupted. Then Sim will send $(\text{TRYIMPERSONATE}, \text{sid}, \text{ssid}, \mathcal{C}, \mathbf{b}')$ to $\mathcal{F}_{2\text{FA}}^{\text{out}}$ and get back $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, d)$ as $\mathcal{F}_{2\text{FA}}^{\text{out}}$'s answer to the corrupted server. Sim creates a fake probe template so that the distance to the earlier fake reference template is exactly d and encrypts and signs the message with the corresponding fake keys. Sim then gives $m = (\text{auth}, \text{rid}, c, \sigma)$ to \mathcal{Z} .

In both worlds rid is uniformly random and stays the same over multiple calls to TRYIMPERSONATE . The encryption and signature keys are identically distributed and also stay the same for multiple calls to TRYIMPERSONATE . The only difference is that the ciphertext c in the real world is the encryption of $\widehat{\mathbf{b}}'$, whereas in the ideal world it is the encryption of the fake probe template $\widehat{\mathbf{b}}'$. In Lemma 1 we show that if \mathcal{Z} can distinguish between the real and the ideal world, there is a reduction which breaks the fh-IND-security of the FE scheme.

Lemma 1. *If \mathcal{Z} can distinguish between a key $\text{sk}_{\mathbf{b}}$ in the real world and the ideal world, or between a ciphertext c in the real world and the ideal world, then there is a reduction \mathcal{B} that wins the fh-IND-security experiment of the FE scheme.*

Proof sketch. We use a hybrid argument over an upper bound on the number of honest clients l . Let H_i be the execution in which the first i honest clients use keys and ciphertexts as produced by the simulator. The other clients are still executed as in the real world. In a bit more detail, in H_i , for honest clients $\{1, \dots, i\}$, whenever an enrolment-message is delivered to a corrupted server, \mathcal{Z} gets the output of the “on $(\text{ENROL}, \text{sid}, \text{ssid}, \text{rid})$ from $\mathcal{F}_{2\text{FA}}^{\text{out}}$ ”-interface of the simulator. Whenever an authentication-message of one of the first i clients is delivered to a corrupted server, \mathcal{Z} gets the output of the “on $(\text{AUTH}, \text{sid}, \text{ssid}, \text{rid}, d)$ from $\mathcal{F}_{2\text{FA}}^{\text{out}}$ ”-interface of the simulator. For clients $\{i + 1, \dots, l\}$, \mathcal{Z} gets the output of

the real clients ENROL (resp. AUTH) interface. So in H_0 all secret keys $\text{sk}_{\mathbf{b}}$ and ciphertexts c are produced as in the real world, whereas in H_l all secret keys and ciphertexts are produced as in the ideal world. An environment that is able to tell apart the real world from the ideal world by distinguishing between the real and simulated FE keys or ciphertexts, is also able to distinguish between H_0 and H_l . Therefore, there must exist $i \in \{1, \dots, l\}$ such that \mathcal{Z} can distinguish between H_{i-1} and H_i . We give a reduction \mathcal{B} that wins the fh-IND-security experiment, given a distinguisher \mathcal{D} for H_{i-1} and H_i :

When \mathcal{Z} calls the “(ENROL, sid, ssid, \mathbf{b})”-interface of the i -th honest client, \mathcal{B} takes the public parameters from the fh-IND FE security experiment and asks a QKeyGen($\mathbf{b}, \widehat{\mathbf{b}}$) query, where \mathbf{b} is the encoding of \mathbf{b} and $\widehat{\mathbf{b}} = \text{encodeRef}(\text{chooseFakeRef}())$ is the encoding of the fake reference template. \mathcal{B} receives back the functional decryption key sk and gives this as part of the enrolment-message to the corrupted server and thereby to \mathcal{Z} . When \mathcal{Z} calls the “(AUTH, sid, ssid, \mathbf{b}')”-interface of the i -th honest client, \mathcal{B} asks a QEnc($\mathbf{b}', \widehat{\mathbf{b}'}$) query, where \mathbf{b}' is the encoding of \mathbf{b}' and $\widehat{\mathbf{b}'}$ is the encoding of the fake probe template that the simulator would have chosen via $\text{chooseFakeProbe}(\cdot, \cdot)$. \mathcal{B} receives back the ciphertext c and gives this as part of the authentication-message to the corrupted server and thereby to \mathcal{Z} .

When the experiment’s bit $b = 0$, then \mathcal{B} gets the secret key and ciphertexts for the real biometric templates, whereby \mathcal{B} perfectly simulates H_{i-1} . When the experiment’s bit $b = 1$, then \mathcal{B} gets the secret key and ciphertexts for the fake biometric templates chosen by the simulator, whereby \mathcal{B} perfectly simulates H_i .

Lemma 2. *There is a reduction \mathcal{B} that wins the EUF-CMA game if the environment manages to get to the else-case in line 34 of the simulator with a valid signature σ .*

Proof sketch. The general idea is that \mathcal{B} runs the simulator’s code, but whenever the simulator would create a signature keypair, or sign a message, \mathcal{B} instead uses its challenger to get the keypair or signature.

A bit more in detail, \mathcal{B} will guess a client \mathcal{C}^* . When Sim creates a keypair for that client in line 60 in Fig. 10), \mathcal{B} will get the public key from its EUF-CMA challenger. Whenever Sim would create a signature under the corresponding secret key (e.g. in line 77 in Fig. 10), \mathcal{B} asks a signing query to their challenger and uses the response as the signature that Sim would have created. When \mathcal{B} gets a “(SEND, (sid, ssid), $\mathcal{S}, m = (\text{auth}, \text{rid}, c, \sigma)$)” instruction from \mathcal{Z} with a valid signature σ (relative to the pk associated with rid), and gets to the else-case in line 34 in Fig. 9), \mathcal{B} outputs $((\text{sid}, \text{ssid}, \text{rid}, c), \sigma)$ as forgery to its EUF-CMA challenger.

Now let us argue that this is indeed a valid forgery. First, observe that since \mathcal{B} came to the else-case, it does not have an entry in table T_2 , which means that the message did not belong to an adversarially enrolled client and thereby pk was not chosen by \mathcal{Z} , but by \mathcal{B} ’s EUF-CMA challenger. Second, since \mathcal{B} came to the else-case, it also does not have a matching entry in table T_5 , which means, it did not ask a signing query for $(\text{sid}, \text{ssid}, \text{rid}, c)$ to its challenger in response to a

TRYIMPERSONATE instruction. Therefore, $((\text{sid}, \text{ssid}, \text{rid}, c), \sigma)$ constitutes a valid forgery and \mathcal{B} wins the EUF-CMA game.

References

1. Agrawal, S., Badrinarayanan, S., Mohassel, P., Mukherjee, P., Patranabis, S.: BETA: biometric-enabled threshold authentication. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12711, pp. 290–318. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_11
2. Agrawal, S., et al.: Game-set-MATCH: using mobile devices for seamless external-facing biometric matching. In: Ligatti, J., et al. (eds.) ACM CCS 2020, pp. 1351–1370. ACM Press, November 2020. <https://doi.org/10.1145/3372297.3417287>
3. Bauspieß, P., et al.: Post-Quantum secure two-party computation for iris biometric template protection. In: IEEE International Workshop on Information Forensics and Security (WIFS), pp. 1–6. IEEE (2020)
4. Bringer, J., Chabanne, H., Patey, A.: Privacy-preserving biometric identification using secure multiparty computation: an overview and recent trends. IEEE Sig. Process. Mag. **30**(2), 42–52 (2013)
5. Cachet, C., et al.: Proximity searchable encryption for biometrics. Cryptology ePrint Archive, Report 2020/1174 (2020). <https://eprint.iacr.org/2020/1174>
6. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000). <https://eprint.iacr.org/2000/067>
7. Cheon, J.H., Kim, D., Kim, D., Lee, J., Shin, J., Song, Y.: Lattice-based secure biometric authentication for hamming distance. In: Baek, J., Ruj, S. (eds.) ACISP 2021. LNCS, vol. 13083, pp. 653–672. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90567-5_33
8. Daugman, J.: How iris recognition works. In: The Essential Guide to Image Processing, pp. 715–739. Elsevier, Amsterdam (2009)
9. Deng, J., et al.: Arcface: additive angular margin loss for deep face recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4690–4699 (2019)
10. Dupont, P.-A., Hesse, J., Pointcheval, D., Reyzin, L., Yakubov, S.: Fuzzy password-authenticated key exchange. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 393–424. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_13
11. Erwig, A., Hesse, J., Ortl, M., Riahi, S.: Fuzzy asymmetric password-authenticated key exchange. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 761–784. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_26
12. FIDO specifications. <https://fidoalliance.org/specifications/> (visited on 01/12/2023)
13. Gardham, D., Manulis, M., Drăgan, C.C.: Biometric-authenticated searchable encryption. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 2020. LNCS, vol. 12147, pp. 40–61. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57878-7_3
14. Gunasinghe, H., Bertino, E.: PrivBioMTAuth: privacy preserving biometrics-based and user centric protocol for user authentication from mobile phones. IEEE Trans. Inf. Forensics Secur. **13**(4), 1042–1057 (2017)

15. Ibarrodo, A., Chabanne, H., Önen, M.: Practical privacy-preserving face identification based on function-hiding functional encryption. In: Conti, M., Stevens, M., Krenn, S. (eds.) CANS 2021. LNCS, vol. 13099, pp. 63–71. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92548-2_4
16. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric pake protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 456–486. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_15
17. Jarecki, S., Krawczyk, H., Shirvanian, M., Saxena, N.: Two-factor authentication with end-to-end password security. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. LNCS, vol. 10770, pp. 431–461. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76581-5_15
18. Kim, S., Lewi, K., Mandal, A., Montgomery, H., Roy, A., Wu, D.J.: Function-hiding inner product encryption is practical. In: Catalano, D., De Prisco, R. (eds.) SCN 2018. LNCS, vol. 11035, pp. 544–562. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98113-0_29
19. Kolberg, J., et al.: Template protection based on homomorphic encryption: computationally efficient application to iris-biometric verification and identification. In: IEEE International Workshop on Information Forensics and Security (WIFS), pp. 1–6. IEEE (2019)
20. Liu, W., et al.: SpheroFace: deep hypersphere embedding for face recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 212–220 (2017)
21. Marc, T., Stopar, M., Hartman, J., Bizjak, M., Modic, J.: Privacy-enhanced machine learning with functional encryption. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019. LNCS, vol. 11735, pp. 3–21. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29959-0_1
22. Rabin, M.O., Shallit, J.O.: Randomized algorithms in number theory. *Commun. Pure Appl. Math.* **39**(S1), S239–S256 (1986)
23. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: a unified embedding for face recognition and clustering. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 815–823 (2015)
24. Wang, M., et al.: Biometrics-authenticated key exchange for secure messaging. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. ACM Press, pp. 2618–2631, November 2021. <https://doi.org/10.1145/3460120.3484746>
25. Zhou, K., Ren, J.: PassBio: privacy-preserving user-centric biometric authentication. *IEEE Trans. Inf. Forensics Secur.* **13**(12), 3050–3063 (2018)



Private Information Retrieval with Result Verification for More Servers

Pengzhen Ke and Liang Feng Zhang^(✉)

ShanghaiTech University, Shanghai, China
{kepzh, zhanglf}@shanghaitech.edu.cn

Abstract. Private information retrieval (PIR) allows a client to retrieve an element from a database without revealing which element is downloaded to the database servers. PIR protocols with unconditional privacy and sublinear (in n) communication complexity can be constructed assuming multiple honest-but-curious servers. This assumption however cannot be guaranteed in many real life scenarios such as using cloud servers as database servers. We consider an multi-server information-theoretic PIR with result verification (PIR-RV) model where the client can detect the existence of malicious servers even if only one server is honest. We construct a t -private k -server PIR-RV protocol for arbitrary $k \geq 2$ and $1 \leq t < k$, and show its security for $2 \leq k \leq 5$. The protocol's communication complexity is $O(\frac{k^2}{t}(\frac{nk}{t})^{1/(\lfloor(2k-1)/t\rfloor-1)} \log p)$, where p is the size of finite field.

Keywords: Cryptography · Private Information Retrieval · Malicious Servers · Security

1 Introduction

Private information retrieval (PIR) [6] allows a client to retrieve an element from a database without revealing which element is downloaded to the database servers. The communication complexity of a PIR protocol is the total number of bits that the client must exchange in order to retrieve one bit from the database. PIR is a fundamental privacy-preserving primitive in cryptography and has applications in many systems such as private media browsing [12], metadata-private messaging [1] and location-based services for smartphones [14, 20].

A trivial PIR protocol requires the client to download the whole database from a server and incurs a prohibitive communication complexity that is linear in the size of the database. Chor et al. [6] showed that if there is only one server and the perfect privacy of the retrieval index is required, then the linear communication complexity is unavoidable. A *multi-server* PIR model [2–4, 8, 9, 18, 21] is necessary if both perfect privacy and the sublinear communication complexity are required. In a *multi-server* PIR model, the database is replicated among multiple servers and the client retrieves a database element by querying every server once, such that each individual server learns no information about the retrieval index. A t -private k -server PIR protocol [18] provides the stronger

security guarantee that the retrieval index is perfectly private for any t colluding servers.

While the early PIR protocols always assume *honest-but-curious* servers that strictly follow the protocol's specifications but try to learn the retrieval index, recently a lot of efforts [5, 7, 10, 13, 15, 19, 22–25] have been made to deal with *malicious* servers that may collude and provide wrong answers to the client, in order to deceive the client into reconstructing an incorrect value. The PIR protocols that tolerate malicious servers have particular interest in the modern age of cloud computing because they allow the PIR servers to be implemented by the untrustworthy cloud services, i.e., outsourcing the PIR servers' computations to the cloud.

Byzantine-Robust PIR (BRPIR). Informally, BRPIR protocols [5, 7, 10, 15, 19, 23] not only allow the client to learn the correct value of the retrieved database element in the presence of a limited number of malicious servers but also allow the client to figure out which servers are cheating (see Table 1). A t -private v -Byzantine-robust k -out-of- ℓ PIR ((t, v, k, ℓ) -BRPIR) protocol ensures that if at least k out of the ℓ servers answer the client's queries and at most v of the k answering servers are malicious (called *Byzantine* servers) and return incorrect answers, then the client is able to both reconstruct the correct value of the database element and figure out the incorrect answers. Beimel and Stahl [5] constructed such systems for $v \leq t \leq k/3$. The BRPIR protocol of Yang et al. [19] allows $v \leq t < k/2$. The BRPIR protocols in both [5, 19] require an *exponential time* reconstructing algorithm. Goldberg [10] improved the robustness with protocols that satisfy $0 < t < k$ and $v < k - \lfloor \sqrt{kt} \rfloor$. Devet and Goldberg [7] got further improvements with $0 < t < k$ and $v < k - t - 1$. Both [7, 10] run in *polynomial time* but only result in a list of candidates for the value of the retrieved database element. Kurosawa [15] proposed polynomial-time reconstructing algorithms for the BRPIR protocol in [5]. Zhang et al. [23] proposed a 1-private BRPIR protocol with both a polynomial time reconstructing algorithm and a communication complexity lower than [15].

Verifiable PIR (VPIR). The VPIR protocols [22, 24] are constructed in either the multi-server model or the single-server model such that the wrong answers provided by the malicious servers will be detected and identified (but there is no guarantee of recovering the correct element, or even a small list of candidates). Compared with BRPIR, VPIR has higher efficiency and lower communication complexity. In particular, the k -server VPIR protocol's communication complexity is comparable to the best existing k -server PIR protocols for honest-but-curious servers. Besides, VPIR protocols allow all the servers to be malicious (*Byzantine* in BRPIR, see Table 1). The security of VPIR protocols in both [22, 24] is based on cryptographic assumptions. Therefore, they do not provide *information-theoretic* security. More importantly, VPIR protocols require a *trusted third party*, called the data owner, to preprocess the database and set up the public keys for verification. Another drawback of these protocols is that to facilitate verification, both the client and the server(s) in these protocols have to

do a lot of public-key operations such as pairing computations or lattice-related computations.

Cheating Detectable PIR (CD-PIR). The CD-PIR protocol [25] allows the client to detect the wrong answers provided by the malicious servers (but there is no guarantee of identifying which servers are malicious, see Table 1). The k -server CD-PIR also has comparable communication complexity to the best existing k -server PIR protocols. Also, CD-PIR still works even if all of the servers are malicious. Compared with VPIR, CD-PIR does not have to do the costly public-key operations. Similar as VPIR, however, CD-PIR also requires the data owner to be honest (as a *trusted third party*) to preprocess the database and set up the keys for verification. The security of CD-PIR is not *information-theoretic* but based on the collision resistance of hash functions.

PIR with Result Verification (PIR-RV). The PIR-RV protocol [13] is constructed only in a 2-server model. Similar as CD-PIR, PIR-RV protocol allows the client to detect the wrong answers, provides *information-theoretic* security and requires no public-key operation (see Table 1). Compared with CD-PIR, PIR-RV protocol has only two participants: the client and the servers. The security of PIR-RV relies on the assumption that one of the servers is honest. Such assumption is much more practical compared to the assumption that there is an honest data owner playing the role of a trusted third party (used in VPIR and CD-PIR).

Compared with BRPIR, VPIR, and CD-PIR, if we restrict to protocols using the same number of servers, the PIR-RV protocol from [13] is much more efficient, either in terms of communication complexity or computational complexity. However, a main limitation of [13] is that it only gives a protocol for 2 servers. It's very interesting to extend their PIR-RV study to k -server protocols for any $k \geq 2$ since more servers may indicate higher efficiency. However, such an extension seems highly nontrivial, if we consider k -server PIR-RV protocols whose result verification property holds in the presence of as many as $k - 1$ malicious servers. The main reason is that the very specific proof techniques of [13] for security cannot be easily extended to the $k > 2$ cases.

1.1 Our Contribution

In this paper, we extend the PIR-RV study of [13] for more than 2 servers. Specifically, we make the following contributions.

- We extend the 2-server PIR-RV model to k -server PIR-RV model for arbitrary $k \geq 2$. Our model requires that the client should be able to detect the existence of cheating servers even if only one of the servers is honest. Therefore, our k -server PIR-RV protocol has $k - 1$ malicious servers tolerance. In fact, it is impossible for a protocol to get k malicious servers tolerance if it is free of trusted third-party. We will prove this conclusion in Sect. 3.
- We construct a t -private k -server PIR-RV protocol Γ for any $k \geq 2$ and $1 \leq t < k$. As the security proofs are quite challenging, we show the security of

our k -server PIR-RV protocols for $2 \leq k \leq 5$. The communication complexity of our t -private k -server PIR-RV protocol is

$$O\left(\frac{k^2}{t} \left(\frac{nk}{t}\right)^{1/(\lfloor(2k-1)/t\rfloor-1)} \log p\right).$$

For $k = 2$ and $t = 1$, it is consistent with the $O(n^{1/2} \log p)$ communication complexity of the 2-server PIR-RV protocol of [13].

Table 1. Comparisons between the t -private k -server models of PIR, BRPIR, VPIR, CD-PIR and PIR-RV. (We denote by ‘IT’ *information-theoretic* security and by ‘C’ *computational* security.)

	PIR	BRPIR [10]	VPIR [22]	CD-PIR [25]	PIR-RV (This paper)
Privacy	✓	✓	✓	✓	✓
Security	×	IT	C	C	IT
Result Verification	×	✓	✓	✓	✓
Malicious Servers Identification	×	✓	✓	×	×
Free of Trusted Third-Party	✓	✓	×	×	✓
Free of Public-Key Operation	✓	✓	×	✓	✓
Malicious Servers Tolerance	×	$k - t - 1$	k	k	$k - 1$

1.2 Paper Organization

In Sect. 2, we introduce the notation and necessary tools for constructing our PIR-RV protocols. In Sect. 3, we show a k -server PIR-RV model for any $k \geq 2$ and define the correctness, privacy, security for PIR-RV protocols. We give the construction of our PIR-RV protocol Γ for $k \geq 2$ and $1 \leq t < k$ in Sect. 4 and show the security proof for $2 \leq k \leq 5$. Finally, Sect. 5 contains our concluding remarks.

2 Preliminaries

2.1 Notation

For any positive integer n , we denote $[n] = \{1, 2, \dots, n\}$ and $\{a_j\}_{j \in [n]} = \{a_1, a_2, \dots, a_n\}$. For any prime p , we denote by \mathbb{F}_p the finite field of p elements, denote $\mathbb{F}_p^* = \mathbb{F}_p \setminus \{0\}$ and denote by \mathbb{F}_p^n the n -dimensional vector space over \mathbb{F}_p . For any vector $V \in \mathbb{F}_p^n$ and any $i \in [n]$, we denote by $V[i]$ the i -th element of V . For any two vectors $V_1, V_2 \in \mathbb{F}_p^n$, we denote by $\langle V_1, V_2 \rangle$ the inner product of V_1 and V_2 . For any function $f(\lambda)$ and any positive integer j , we denote by $f^{(j)}(\lambda)$ the j -order derivative of $f(\lambda)$. In particular, when $j = 1$, we denote $f'(\lambda) = f^{(1)}(\lambda)$. We denote by $\neg E$ the complement of any event E . For any

two polynomials $g(\lambda)$ and $h(\lambda)$, the notion $g(\lambda) \mid h(\lambda)$ means that $g(\lambda)$ divides $h(\lambda)$ or $h(\lambda)$ is a *multiple* of $g(\lambda)$, i.e., there exists a polynomial $f(\lambda)$ such that $h(\lambda) = f(\lambda)g(\lambda)$. The notion $g(\lambda) \nmid h(\lambda)$ means that $g(\lambda)$ does not divide $h(\lambda)$ and $h(\lambda)$ is not a multiple of $g(\lambda)$.

2.2 Constant Weight Code

For integers $m, d, w > 0$, a binary (m, d, w) -constant weight code is a code with codeword length m , minimum Hamming distance d and Hamming weight w . Let $B(m, d, w)$ be the maximum size of a binary (m, d, w) -constant weight code. Graham and Sloane [11] established a lower bound of $B(m, 4, w)$ by giving an explicit construction of $(m, 4, w)$ -constant weight code.

Theorem 1. (Graham and Sloane [11]) *For any positive integers m, w such that $m \geq w$, $B(m, 4, w) \geq \binom{m}{w}/m$.*

2.3 Cramer's Rule

Consider a system of n linear equations in n unknowns:

$$Ax = \mathbf{b},$$

where A is a nonsingular $n \times n$ coefficient matrix and $x = [x[1] \cdots x[n]]^\top$ is the column vector of n unknowns. Cramer's rule [16] states that the equation system has a unique solution given by

$$x[i] = \frac{|A_i|}{|A|}, \quad 1 \leq i \leq n,$$

where A_i is the matrix formed by replacing the i -th column of A by the column vector \mathbf{b} .

2.4 Woodruff-Yekhanin PIR

The t -private k -server PIR-RV protocol in this paper is based on the t -private k -server Woodruff-Yekhanin PIR protocol [18], which encodes the database as a multivariate polynomial and uses Shamir's secret sharing scheme [17] to protect the privacy of the client's retrieval index.

A k -server PIR is a communication protocol between a *client* and k *servers* $\{\mathcal{S}_j\}_{j \in [k]}$, where each server has a copy of a database $DB = (DB[1], \dots, DB[n]) \in \{0, 1\}^n$ and the client wants to retrieve $DB[i]$ for a private index $i \in [n]$, by interacting with the servers. In such a protocol, the client sends a query to each server, receives an answer from each server, and finally reconstructs $DB[i]$ from the k answers. Such a protocol is said to be t -private if any t out of the k queries give no information about i .

To retrieve an item from a database $DB = (DB[1], \dots, DB[n]) \in \{0, 1\}^n$, Woodruff and Yekhanin let $w = \lfloor (2k - 1)/t \rfloor$ and choose an integer m such that $\binom{m}{w} \geq n$. Then they construct a *public* 1-to-1 index encoding $E : [n] \rightarrow \{0, 1\}^m$ that encodes any $i \in [n]$ as a codeword of a binary $(m, 2, w)$ -constant weight code. The database DB is encoded as

$$F(z) = F(z[1], \dots, z[m]) = \sum_{j=1}^n DB[j] \prod_{\ell: E(j)[\ell]=1} z[\ell], \tag{1}$$

a homogeneous m -variate polynomial of degree w such that

$$F(E(i)) = DB[i]$$

for all $i \in [n]$. The client reduces the problem of privately retrieving $DB[i]$ from k servers to the problem of privately evaluating $F(E(i))$ with the k servers. To this end, a prime $p > k$ is chosen and F is interpreted as a polynomial over the finite field \mathbb{F}_p . For each $j \in [k]$, the j -th server is associated with a nonzero field elements λ_j . In particular, the field elements $\{\lambda_j\}_{j \in [k]}$ are distinct and can be made public. To retrieve $DB[i]$, the client chooses t vectors $V_1, V_2, \dots, V_t \in \mathbb{F}_p^m$ uniformly and generates a polynomial of degree t ,

$$G(\lambda) = E(i) + \sum_{s=1}^t \lambda^s V_s. \tag{2}$$

The client then lets $\mathbf{aux} = (\{\lambda_j\}_{j \in [k]}, \{V_j\}_{j \in [t]})$. For each $j \in [k]$, the client sends $Q_j = G(\lambda_j)$ to the j -th server. The j -th server then computes $A_j[1] = F(Q_j)$ and

$$A_j[\ell + 1] = \left. \frac{\partial F(z)}{\partial z[\ell]} \right|_{Q_j}$$

for all $\ell \in [m]$, and returns an answer $A_j = (A_j[1], \dots, A_j[m + 1])$ to the client. To reconstruct the database element $DB[i]$, the client depends on the following lemma.

Lemma 1. (Woodruff and Yekhanin [18]) *Let $\{\lambda_j\}_{j \in [k]}$ be a set of k distinct non-zero elements the finite field \mathbb{F}_p . Let $\{u_j\}_{j \in [k]}$ and $\{v_j\}_{j \in [k]}$ be two subsets of \mathbb{F}_p^k . There exists at most one polynomial $f(\lambda) \in \mathbb{F}_p[\lambda]$ of degree $\leq 2k - 1$ such that $f(\lambda_j) = u_j$ and $f'(\lambda_j) = v_j$ for all $j \in [k]$.*

Specifically, the client will consider the univariate polynomial

$$f(\lambda) = F(G(\lambda)). \tag{3}$$

On one hand, it learns that $f(\lambda_j) = A_j[1]$ for all $j \in [k]$. On the other hand, by the chain rule, it learns that

$$f'(\lambda_j) = \sum_{\ell=1}^m \left. \frac{\partial F(z)}{\partial z[\ell]} \right|_{Q_j} \cdot G'(\lambda_j)[\ell] = \sum_{\ell=1}^m A_j[\ell + 1] \cdot G'(\lambda_j)[\ell].$$

By Lemma 1, the client can interpolate the polynomial $f(\lambda)$ of degree wt ($\leq 2k - 1$) with the $2k$ values. Finally, the client learns that

$$DB[i] = F(E(i)) = f(0).$$

Since $\{V_j\}_{j \in [t]}$ are chosen uniformly, for any $T \subseteq [k]$ with $|T| \leq t$, the set $\{G(\lambda_h)\}_{h \in T} = \{E(i) + \sum_{j=1}^t \lambda_h^j V_j\}_{h \in T}$ discloses no information about $E(i)$. The statement is implied by the properties of Shamir secret sharing scheme [17]. Therefore, the protocol is t -private. Clearly, the client sends a length- m vector in \mathbb{F}_p to each server and each server returns a length- $(m + 1)$ vector in \mathbb{F}_p to the client. The communication complexity of this t -private k -server PIR protocol is $O(km \log p)$. Recall that $m = O(wn^{1/w})$. If the prime p is chosen such that $k < p \leq 2k$, then the communication complexity of this t -private k -server PIR protocol is $O(\frac{k^2 \log k}{t} n^{1/\lfloor \frac{2k-1}{t} \rfloor})$.

3 The PIR-RV Model

In this section, we extend the 2-server PIR-RV model in [13] to a k -server model for any $k \geq 2$. Our k -server PIR-RV model involves two kinds of participants: a *client* and k *servers* $\{S_j\}_{j \in [k]}$. Each server has a copy of a database $DB = (DB[1], \dots, DB[n]) \in \{0, 1\}^n$. The client has an index $i \in [n]$ and wants to privately retrieve the correct value of $DB[i]$ from the k servers. The difference between PIR-RV and PIR is that the servers in PIR-RV could be malicious and try to persuade the client to accept an incorrect result. The client in a PIR-RV needs to verify if the reconstructed result is correct or not.

Definition 1 (PIR-RV). A k -server PIR-RV protocol $\Gamma = (\text{Que}, \text{Ans}, \text{Rec})$ is a triple of algorithms, which can be described as follows:

- $(\{Q_j\}_{j \in [k]}, \text{aux}) \leftarrow \text{Que}(n, i)$: This is a randomized querying algorithm for the client. It takes the database size n and a retrieval index $i \in [n]$ as input, and outputs k queries $\{Q_j\}_{j \in [k]}$, along with an auxiliary information aux . For each $j \in [k]$, the query Q_j will be sent to the server S_j . The auxiliary information aux will be used by the client in the reconstructing algorithm.
- $A_j \leftarrow \text{Ans}(DB, Q_j)$: This is a deterministic answering algorithm for the server S_j ($j \in [k]$). It takes the database DB and the query Q_j as input, and outputs an answer A_j .
- $\{DB[i], \perp\} \leftarrow \text{Rec}(i, \{A_j\}_{j \in [k]}, \text{aux})$: This is a deterministic reconstructing algorithm for the client. It uses the retrieval index i , the answers $\{A_j\}_{j \in [k]}$ and the auxiliary information aux to reconstruct $DB[i]$. The output of this algorithm is expected to be the correct value of $DB[i]$ (when all answers are correct) or a special symbol \perp (which indicates that at least one of the answers is incorrect).

Same as the 2-server PIR-RV model in [13], a protocol Γ in our k -server PIR-RV model should satisfy not only the properties of correctness and t -privacy, but also the property of *security against cheating servers*.

The k -server PIR-RV protocol Γ is said to be *correct* if the reconstructing algorithm Rec always outputs the correct value of $DB[i]$ when all servers are honest.

Definition 2 (Correctness). *The k -server PIR-RV protocol Γ is correct if for any n , any $DB \in \{0, 1\}^n$, any $i \in [n]$, any $(\{Q_j\}_{j \in [k]}, \text{aux}) \leftarrow \text{Que}(n, i)$, it holds that $\text{Rec}(i, \{\text{Ans}(DB, Q_j)\}_{j \in [k]}, \text{aux}) = DB[i]$.*

The k -server PIR-RV protocol Γ is said to be (unconditionally) t -private if no collusion of up to t servers can learn any information about the client's retrieval index i .

Definition 3 (t -Privacy). *The k -server PIR-RV protocol Γ is t -private if for any n , any $i_1, i_2 \in [n]$ and any set $T \subseteq [k]$ where $|T| \leq t$, the distribution of $\text{Que}_T(n, i_1)$ and $\text{Que}_T(n, i_2)$ are identical, where Que_T denotes the concatenation of the j -th output of Que for all $j \in T$.*

Regarding the security of the PIR-RV protocol, we observe that at most $k - 1$ malicious servers can be tolerated. Otherwise, the client with input $i \in [n]$ will be easily deceived into outputting a value $\notin \{DB[i], \perp\}$. In fact, when all of the k servers are malicious and colluding with each other, they may replace the database DB with a new database DB' such that $DB'[j] \neq DB[j]$ for all $j \in [n]$ and execute the rest of the protocol honestly. Then the client will output $DB'[i] \notin \{DB[i], \perp\}$.

Intuitively, the k -server PIR-RV protocol Γ is said to be *secure* if no collusion of up to $k - 1$ servers can cause the client with input i to output a value $\notin \{DB[i], \perp\}$, by providing wrong answers. In our k -server PIR-RV model, we require that the collusion of any $k - 1$ servers should be tolerated. To define the security, we extend the security experiment for 2-server PIR-RV model in [13] to the k -server case for any $k \geq 2$. In our security experiment $\text{EXP}_{\text{Adv}, \Gamma}^{\text{Ver}}(n, DB, i, T)$ (Fig. 1), an adversary Adv controls up to $k - 1$ malicious servers $\{\mathcal{S}_j\}_{j \in T}$, knows both the database DB and the retrieval index i , receives the queries $\{Q_j\}_{j \in T}$, and crafts the answers $\{\hat{A}_j\}_{j \in T}$ for the client. We say that the protocol is ϵ -secure if the client will not output a value $\notin \{DB[i], \perp\}$, except with probability $\leq \epsilon$.

Definition 4 (Security). *The k -server PIR-RV protocol Γ is ϵ -secure if for any adversary Adv , any $T \subsetneq [k]$, any n , any $DB \in \{0, 1\}^n$, and any $i \in [n]$, $\Pr[\text{EXP}_{\text{Adv}, \Gamma}^{\text{Ver}}(n, DB, i, T) = 1] \leq \epsilon$.*

Definition 5 (Communication complexity). *The communication complexity of a k -server PIR-RV protocol Γ for binary database of size n , denoted by $\text{CC}_\Gamma(n, k)$, is the number of bits communicated between the client and all servers, maximized over the choices of $DB \in \{0, 1\}^n$ and $i \in [n]$, i.e., $\text{CC}_\Gamma(n, k) = \max_{DB, i} (\sum_{j=1}^k (|Q_j| + |A_j|))$.*

1. The challenger generates $(\{Q_j\}_{j \in [k]}, \text{aux}) \leftarrow \text{Que}(n, i)$ and sends $\{Q_j\}_{j \in T}$ to the adversary Adv .
2. The adversary Adv sends the answers $\{\hat{A}_j\}_{j \in T}$ to the challenger.
3. The challenger computes $\hat{A}_j \leftarrow \text{Ans}(DB, Q_j)$ for all $j \in [k] \setminus T$.
4. The challenger runs $\text{Rec}(i, \{\hat{A}_j\}_{j \in [k]}, \text{aux})$ and gets an output \hat{y} .
5. If $\hat{y} \notin \{DB[i], \perp\}$, outputs 1; otherwise outputs 0.

Fig. 1. The security experiment $\text{EXP}_{\text{Adv}, T}^{\text{Ver}}(n, DB, i, T)$.

4 *k*-Server PIR-RV Protocol

4.1 Overview

In Woodruff-Yekhanin PIR [18], the index encoding E maps the indices in $[n]$ to the codewords of a binary $(m, 2, w)$ -constant weight code, where $w = \lfloor (2k-1)/t \rfloor$. Ke and Zhang [13] proposed a 1-private 2-server PIR-RV protocol based on the 1-private 2-server Woodruff-Yekhanin PIR by replacing the binary $(m, 2, 3)$ -constant weight code with a binary $(m, 4, 3)$ -constant weight code and keeping λ_1 and λ_2 secret from the servers. In this section, we generalize the construction of [13] to construct the t -private k -server PIR-RV protocol for any $k \geq 2$ and $1 \leq t < k$ by replacing the binary $(m, 2, w)$ -constant weight code with a binary $(m, 4, w)$ -constant weight code. The parameters $\{\lambda_j\}_{j \in [k]}$ in our PIR-RV protocol are chosen privately and kept secret from the servers.

In a binary $(m, 4, w)$ -constant weight code, the Hamming distance between any two codewords is at least 4. Therefore, the index encoding E in our PIR-RV protocol satisfies the following property: for any distinct $i, j \in [n]$, there exist distinct $\ell_1, \ell_2 \in [m]$ such that $E(j)[\ell_1] = E(j)[\ell_2] = 1$ and $E(i)[\ell_1] = E(i)[\ell_2] = 0$. Then we get

$$\begin{aligned} & \lambda \left| \left(E(i) + \sum_{s=1}^t \lambda^s V_s \right) [\ell_1], \right. \\ & \left. \lambda \left| \left(E(i) + \sum_{s=1}^t \lambda^s V_s \right) [\ell_2], \right. \right. \\ & \left. \lambda^2 \left| \prod_{\ell: E(j)[\ell]=1} \left(E(i) + \sum_{s=1}^t \lambda^s V_s \right) [\ell]. \right. \right. \end{aligned} \tag{4}$$

That is, the polynomial $\prod_{\ell: E(j)[\ell]=1} (E(i) + \sum_{s=1}^t \lambda^s V_s) [\ell]$ is always a multiple of λ^2 . Besides, we always have that

$$\lambda^2 \nmid \prod_{\ell: E(i)[\ell]=1} \left(E(i) + \sum_{s=1}^t \lambda^s V_s \right) [\ell] \tag{5}$$

for any $i \in [n]$. Let $(f_0, f_1, \dots, f_{wt})$ be the coefficients of the function $f(\lambda)$ in Eq. (3), i.e.,

$$\begin{aligned} f(\lambda) &= \sum_{j=1}^n DB[j] \prod_{\ell: E(j)[\ell]=1} \left(E(i) + \sum_{s=1}^t \lambda^s V_s \right) [\ell] \\ &= \sum_{j=0}^{wt} f_j \lambda^j. \end{aligned}$$

By Eqs. (4) and (5), the terms f_0 and $f_1 \lambda$ in $f(\lambda)$ are completely determined by the polynomial $DB[i] \prod_{\ell: E(i)[\ell]=1} (E(i) + \sum_{s=1}^t \lambda^s V_s) [\ell]$, i.e.,

$$\begin{aligned} f_0 + f_1 \lambda &= f(\lambda) \pmod{\lambda^2} \\ &= \sum_{j=1}^n DB[j] \prod_{\ell: E(j)[\ell]=1} \left(E(i) + \sum_{s=1}^t \lambda^s V_s \right) [\ell] \pmod{\lambda^2} \\ &= DB[i] \prod_{\ell: E(i)[\ell]=1} \left(E(i) + \sum_{s=1}^t \lambda^s V_s \right) [\ell] \pmod{\lambda^2}, \\ f_0 &= DB[i], \\ f_1 &= DB[i] \cdot \langle E(i), V_1 \rangle. \end{aligned}$$

Therefore, we have the result verification equation

$$f_1 = f_0 \cdot \langle E(i), V_1 \rangle, \tag{6}$$

which in particular allows the client to make decision of whether accepting the reconstruction result or not. The coefficients f_0 and f_1 are polynomials of the auxiliary information $\text{aux} = (\{\lambda_j\}_{j \in [k]}, \{V_j\}_{j \in [t]})$ and the servers' answers $\{A_j\}_{j \in [k]}$. If at least one of the k servers is honest, then the collusion of the malicious servers can't learn all information of aux and the honest server's answer. Consequently, it will be hard for the malicious servers to craft incorrect answers that can pass the verification equation. We will prove the security of our result verification method in Sect. 4.2.

4.2 Our Construction

In this section, we show a t -private k -server PIR-RV protocol Γ in Fig. 2 for any $k \geq 2$ and $1 \leq t < k$.

The reconstructing algorithm Rec in our k -server PIR-RV protocol Γ always outputs the correct value of $DB[i]$ when all servers are honest.

Theorem 2. *The k -server PIR-RV protocol Γ is correct.*

Notations.

- $DB = (DB[1], \dots, DB[n])$: a binary database in $\{0, 1\}^n$.
- $w = \lfloor (2k - 1)/t \rfloor$: the degree of the polynomial that encodes DB .
- m : an integer such that there is a binary $(m, 4, w)$ -constant weight code C with $|C| \geq n$.
- $E : [n] \rightarrow C$: A *public* 1-to-1 encoding function.
- $F(z)$: A degree- w homogeneous encoding of the database DB with representation

$$F(z) = F(z[1], \dots, z[m]) = \sum_{j=1}^n DB[j] \prod_{\ell: E(j)[\ell]=1} z[\ell].$$

$(\{Q_j\}_{j \in [k]}, \mathbf{aux}) \leftarrow \text{Que}(n, i)$:

1. Choose a prime $p > k$. Randomly choose $V_1, \dots, V_t \in \mathbb{F}_p^m$ and distinct $\lambda_1, \dots, \lambda_k \in \mathbb{F}_p^*$. Let $\mathbf{aux} = (\{\lambda_j\}_{j \in [k]}, \{V_j\}_{j \in [t]})$.
2. Let

$$G(\lambda) = E(i) + \sum_{s=1}^t \lambda^s V_s.$$

For each $j \in [k]$, $Q_j = G(\lambda_j)$.

3. Output $(\{Q_j\}_{j \in [k]}, \mathbf{aux})$.

$A_j \leftarrow \text{Ans}(DB, Q_j)$:

1. Compute $A_j[1] = F(Q_j)$.
2. For each $\ell \in [m]$,

$$A_j[\ell + 1] = \left. \frac{\partial F(z)}{\partial z[\ell]} \right|_{Q_j}.$$

3. Let $A_j = (A_j[1], \dots, A_j[m + 1])$, output A_j .

$\{DB[i], \perp\} \leftarrow \text{Rec}(i, \{A_j\}_{j \in [k]}, \mathbf{aux})$:

1. Parse $\mathbf{aux} = (\{\lambda_j\}_{j \in [k]}, \{V_j\}_{j \in [t]})$. For each $j \in [k]$, parse $A_j = (A_j[1], \dots, A_j[m + 1])$. Construct the polynomial $G(\lambda) = E(i) + \sum_{s=1}^t \lambda^s V_s$.
2. Interpolate a polynomial $f(\lambda) = \sum_{j=0}^{2k-1} f_j \lambda^j$ of degree $2k - 1$ with $2k$ values

$$f(\lambda_j) = A_j[1], \quad f'(\lambda_j) = \sum_{\ell=1}^m A_j[\ell + 1] \cdot G'(\lambda_j)[\ell], \quad 1 \leq j \leq k.$$

3. If $f_1 = f_0 \cdot \langle E(i), V_1 \rangle$, output $f_0 (= DB[i])$. Otherwise output \perp .

Fig. 2. t -Private k -server PIR-RV protocol Γ .

Proof. The polynomial $F(z)$ in the protocol Γ is a homogeneous m -variate polynomial of degree w , while $G(\lambda)$ is a polynomial of degree t . Let $g(\lambda) = F(G(\lambda))$

be the composite polynomial of degree wt . For each $j \in [k]$, we have that

$$\begin{aligned} g(\lambda_j) &= F(G(\lambda_j)) = F(Q_j) = f(\lambda_j), \\ g'(\lambda_j) &= \sum_{\ell=1}^m \left. \frac{\partial F(z)}{\partial z_\ell} \right|_{Q_j} G'(\lambda_j)[\ell] \\ &= \sum_{\ell=1}^m A_j[\ell + 1]G'(\lambda_j)[\ell] \\ &= f'(\lambda_j). \end{aligned}$$

By Lemma 1, $f(\lambda)$ and $g(\lambda)$ are the same polynomial,

$$\begin{aligned} f(\lambda) = g(\lambda) &= F(G(\lambda)) = \sum_{j=1}^n DB[j] \prod_{\ell: E(j)[\ell]=1} (E(i) + \sum_{s=1}^t \lambda^s V_s)[\ell] \\ f_0 = f(0) &= F(G(0)) = F(E(i)) = DB[i]. \end{aligned}$$

Since the binary $(m, 4, d)$ -constant weight code C has Hamming distance ≥ 4 , for any $j \in [n] \setminus \{i\}$, there exist distinct $\ell_1, \ell_2 \in [m]$ such that $E(j)[\ell_1] = E(j)[\ell_2] = 1$ and $E(i)[\ell_1] = E(i)[\ell_2] = 0$. Then Eqs. (4) and (5) hold for any distinct $i, j \in [n]$. Therefore, Eq. (6) holds, i.e.,

$$\text{Rec}(i, \{\text{Ans}(DB, Q_j)\}_{j \in [k]}, \text{aux}) = f_0 = DB[i].$$

□

No collusion of up to t servers in our k -server protocol Γ can learn any information about the client’s retrieval index i .

Theorem 3. *The k -server PIR-RV protocol Γ is t -private.*

Proof. The proof for the privacy of our PIR-RV protocol is identical to that of Woodruff-Yekhanin PIR [18]. By the property of Shamir’s secret sharing scheme [17], for any $T \subseteq [k]$, where $|T| \leq t$, the set $\{Q_j\}_{j \in T} = \{G(\lambda_j)\}_{j \in T} = \{E(i) + \sum_{s=1}^t \lambda_j^s V_s\}_{j \in T}$ is uniformly distributed over $\mathbb{F}_p^{m \times |T|}$. Therefore, for any n , any distinct $i_1, i_2 \in [n]$ and any set $T \subseteq [k]$ where $|T| \leq t$, the distribution of $\{E(i_1) + \sum_{s=1}^t \lambda_j^s V_s\}_{j \in T}$ and $\{E(i_2) + \sum_{s=1}^t \lambda_j^s V_s\}_{j \in T}$ are identical. □

The security of k -server PIR-RV protocols requires that any collusion of up to $k - 1$ servers cannot deceive the client with input i into reconstructing a value $\notin \{DB[i], \perp\}$. In the following Theorem 4, we show that our k -server PIR-RV protocol Γ (Fig. 2) is secure for $2 \leq k \leq 5$. While this result seems limited, we believe that proving the security of k -server PIR-RV protocols is highly nontrivial. In particular, the proof techniques of [13] cannot be generalized to prove the security of our protocol Γ for more than 2 servers (i.e., $k > 2$). To see this, we note that the security proof of [13] reduces the event that an adversary Adv wins in the security experiment to the event that Adv constructs a nonzero

bivariate polynomial ν such that $\nu(\lambda_1, \lambda_2) = 0$. The proof heavily depends on the fact that $\{\lambda_j\}_{j \in [2]}$ is a uniformly distributed set in Adv 's point of view. In our k -server PIR-RV model, the adversary controls $k - 1$ servers rather than a single server. The proof technique of [13] is no longer applicable because the set $\{\lambda_j\}_{j \in [k]}$ is no longer uniformly distributed in Adv 's point of view.

As a main contribution of this work, we develop new techniques to prove the security of Γ for $2 \leq k \leq 5$. Assume that $T = [k] \setminus \{1\}$ and \mathcal{S}_1 is the only honest server. In our security proof, we use Cramer's rule to represent and generalize the verification equation (Eq. (6)) in the form of a determinant of a matrix. We manage to reduce the event that an adversary Adv wins in the security experiment to the event that Adv constructs a nonzero univariate polynomial ν such that $\nu(\lambda_1) = 0$. We prove that our PIR-RV protocols can resist the adversary who knows $(\{\lambda_j\}_{j \in T}, \{Q_j\}_{j \in T}, \{V_j\}_{j \in [t]}, DB, E(i))$ but λ_1 and Q_1 . Such adversary knows everything that Adv defined in the security experiment $\text{EXP}_{\text{Adv}, \Gamma}^{\text{Ver}}(n, DB, i, T)$ knows. Therefore our PIR-RV protocol can resist Adv and thus it is secure.

Theorem 4. *The k -server PIR-RV protocol Γ is $(4k - 4)/(p - k)$ -secure for $2 \leq k \leq 5$.*

Proof. It suffices to consider an adversary Adv that controls the $k - 1$ servers $\{\mathcal{S}_j\}_{j \in T}$ for $T = [k] \setminus \{1\}$ in the security experiment $\text{EXP}_{\text{Adv}, \Gamma}^{\text{Ver}}(n, DB, i, T)$.

Clearly, λ_1 is uniformly distributed in $\mathbb{F}_p^* \setminus \{\lambda_j\}_{j \in T}$ in Adv 's point of view and Adv has no information about λ_1 . We reduce the event that Adv wins in the security experiment to the event that Adv constructs a nonzero univariate polynomial $\nu(\lambda)$ such that $\nu(\lambda_1) = 0$. For each $j \in [k]$, let $A_j = \text{Ans}(DB, Q_j)$ be the answer obtained by correctly executing the server's answering algorithm. Let $\hat{A}_1 = A_1$ and \hat{A}_j be the answer chosen by Adv for each $j \in T$. Let

$$f(\lambda) = \sum_{j=0}^{2k-1} f_j \lambda^j$$

be the polynomial interpolated with the $2k$ values $\{f(\lambda_j)\}_{j \in [k]}$ and $\{f'(\lambda_j)\}_{j \in [k]}$:

$$f(\lambda_j) = A_j[1], \quad f'(\lambda_j) = \sum_{\ell=1}^m A_j[\ell + 1] \cdot G'(\lambda_j)[\ell], \quad 1 \leq j \leq k. \quad (7)$$

Let

$$\hat{f}(\lambda) = \sum_{j=0}^{2k-1} \hat{f}_j \lambda^j$$

be the polynomial interpolated with the $2k$ values $\{\hat{f}(\lambda_j)\}_{j \in [k]}$ and $\{\hat{f}'(\lambda_j)\}_{j \in [k]}$:

$$\hat{f}(\lambda_j) = \hat{A}_j[1], \quad \hat{f}'(\lambda_j) = \sum_{\ell=1}^m \hat{A}_j[\ell + 1] \cdot G'(\lambda_j)[\ell], \quad 1 \leq j \leq k. \quad (8)$$

Equations (7) and (8) give the following linear equation systems:

$$\underbrace{\begin{bmatrix} f(\lambda_1) \\ f'(\lambda_1) \\ \vdots \\ f(\lambda_k) \\ f'(\lambda_k) \end{bmatrix}}_v = \underbrace{\begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^{2k-1} \\ 0 & 1 & 2\lambda_1 & \dots & (2k-1)\lambda_1^{2k-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_k & \lambda_k^2 & \dots & \lambda_k^{2k-1} \\ 0 & 1 & 2\lambda_k & \dots & (2k-1)\lambda_k^{2k-2} \end{bmatrix}}_M \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{2k-2} \\ f_{2k-1} \end{bmatrix}, \quad (9a)$$

$$\underbrace{\begin{bmatrix} \hat{f}(\lambda_1) \\ \hat{f}'(\lambda_1) \\ \vdots \\ \hat{f}(\lambda_k) \\ \hat{f}'(\lambda_k) \end{bmatrix}}_{\hat{v}} = M \begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \vdots \\ \hat{f}_{2k-2} \\ \hat{f}_{2k-1} \end{bmatrix}. \quad (9b)$$

For brevity, let $Y = \langle E(i), V_1 \rangle$. Adv wins in the security experiment $\text{EXP}_{\text{Adv}, \Gamma}^{\text{Ver}}(n, DB, i, T)$ if and only if the challenger outputs a value not in $\{DB[i], \perp\}$. That is,

$$\hat{f}_0 \neq f_0, \quad (10a)$$

$$\hat{f}_1 = \hat{f}_0 Y. \quad (10b)$$

The event that Adv wins is equivalent to the event that Eqs. (10a) and (10b) both hold. Therefore, we have

$$\text{EXP}_{\text{Adv}, \Gamma}^{\text{Ver}}(n, DB, i, T) = 1 \Leftrightarrow \text{Eq.}(10a) \wedge \text{Eq.}(10b). \quad (11)$$

Since the protocol Γ is correct, the equation

$$f_1 = f_0 Y. \quad (12)$$

always holds. For brevity, let

$$H = (\hat{f}_1 - f_1) - Y(\hat{f}_0 - f_0).$$

Given the fact that Eq. (12) always holds, Eq. (10b) holds if and only if $H = 0$,

$$\text{Eq.}(10b) \Leftrightarrow H = 0. \quad (13)$$

By Cramer's rule (Sect. 2.3), we have that

$$f_0 = |M^{(1)}|/|M|,$$

$$f_1 = |M^{(2)}|/|M|,$$

$$\hat{f}_0 = |\hat{M}^{(1)}|/|M|,$$

$$\hat{f}_1 = |\hat{M}^{(2)}|/|M|,$$

where $M^{(j)}$ and $\hat{M}^{(j)}$ are the matrices formed by replacing the j -th column of M (Eq.(9a)) with the column vectors v (Eq.(9a)) and \hat{v} (Eq.(9b)) respectively for $j = 1, 2$. Therefore, we have

$$\begin{aligned}
 H &= \frac{1}{|M|} ((|\hat{M}^{(2)}| - |M^{(2)}|) - Y (|\hat{M}^{(1)}| - |M^{(1)}|)) \\
 &= \frac{-1}{|M|} \underbrace{\begin{bmatrix} \hat{f}(\lambda_1) - f(\lambda_1) & 1 + Y\lambda_1 & \lambda_1^2 & \dots & \lambda_1^{2k-1} \\ \hat{f}'(\lambda_1) - f'(\lambda_1) & Y & 2\lambda_1 & \dots & (2k-1)\lambda_1^{2k-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{f}(\lambda_k) - f(\lambda_k) & 1 + Y\lambda_k & \lambda_k^2 & \dots & \lambda_k^{2k-1} \\ \hat{f}'(\lambda_k) - f'(\lambda_k) & Y & 2\lambda_k & \dots & (2k-1)\lambda_k^{2k-2} \end{bmatrix}}_N. \tag{15a}
 \end{aligned}$$

Clearly, $H = 0$ if and only if the determinant $|N| = 0$,

$$H = 0 \Leftrightarrow |N| = 0. \tag{16}$$

We consider the determinant $|N|$ as a polynomial of variable λ_1 ,

$$\nu(\lambda_1) = |N| = \sum_{\ell=0}^{4k-4} \nu_\ell \lambda_1^\ell.$$

Then $|N| = 0$ if and only if $\nu(\lambda_1) = 0$, i.e., λ_1 is a zero of the polynomial $\nu(\lambda)$. We next prove that it is impossible for Adv to craft the answers $\{\hat{A}_j\}_{j \in T}$ such that $\nu(\lambda)$ is constantly equal to 0 for $2 \leq k \leq 5$. For brevity, we denote

$$\Delta = \begin{bmatrix} \Delta_1 \\ \Delta_2 \\ \vdots \\ \Delta_{2k-1} \\ \Delta_{2k} \end{bmatrix} = \hat{v} - v = \begin{bmatrix} \hat{f}(\lambda_1) - f(\lambda_1) \\ \hat{f}'(\lambda_1) - f'(\lambda_1) \\ \vdots \\ \hat{f}(\lambda_k) - f(\lambda_k) \\ \hat{f}'(\lambda_k) - f'(\lambda_k) \end{bmatrix}.$$

Since $\hat{A}_1 = A_1$, by Eqs. (7) and (8), we have that

$$\begin{aligned}
 \Delta_1 &= \hat{f}(\lambda_1) - f(\lambda_1) \\
 &= \hat{A}_1[1] - A_1[1] \\
 &= 0, \\
 \Delta_2 &= \hat{f}'(\lambda_1) - f'(\lambda_1) \\
 &= \sum_{\ell=1}^m \hat{A}_1[\ell+1] \cdot G'(\lambda_1)[\ell] - \sum_{\ell=1}^m A_1[\ell+1] \cdot G'(\lambda_1)[\ell] \\
 &= 0.
 \end{aligned} \tag{17}$$

Let N' be the matrix formed by removing the first two rows of N . For each $j \in [2k]$, let $e_j \in \mathbb{F}_p^{2k}$ be the j th standard basis vector. Let

$$\begin{aligned} p(\lambda) &= [0 \ 1 + Y\lambda \ \lambda^2 \ \dots \ \lambda^{2k-1}] \\ &= (1 + Y\lambda)e_2 + \sum_{\ell=3}^{2k} \lambda^\ell e_\ell. \end{aligned}$$

Then we have that

$$\begin{aligned} p(0) &= e_2, \\ p^{(j)}(0) &= \begin{cases} Y e_2, & j = 1; \\ j! e_{j+1}, & 2 \leq j \leq 2k - 1; \\ 0, & j \geq 2k; \end{cases} \end{aligned}$$

By expanding the matrix N (Eq. (15a)) along its first two rows, we have that

$$\begin{aligned} \nu_0 &= \left| \begin{bmatrix} p(0) \\ p^{(1)}(0) \\ N' \end{bmatrix} \right| = 0, \\ j! \nu_j &= \sum_{\ell=0}^j \binom{j}{\ell} \left| \begin{bmatrix} p^{(j-\ell)}(0) \\ p^{(\ell+1)}(0) \\ N' \end{bmatrix} \right|, \quad 1 \leq j \leq 4k - 4. \end{aligned} \tag{18}$$

With Eqs. (17) and (18), we can represent the system of the linear equations of the coefficients $\{\nu_j\}_{j \in [4k-4]}$ in matrix multiplication form:

$$S [\Delta_3 \ \Delta_4 \ \dots \ \Delta_{2k}]^\top = [\nu_1 \ \dots \ \nu_{4k-4}]^\top,$$

where S is a $(4k-4) \times (2k-2)$ matrix whose every entry is a function of $\{\lambda_j\}_{j \in T}$ and Y . For example, when $k = 2$ we have that

$$S = \begin{bmatrix} 6\lambda_2^2 & -2\lambda_2^3 \\ 3\lambda_2^2 Y - 6\lambda_2 & -\lambda_2^3 Y + 3\lambda_2^2 \\ -4\lambda_2 Y & 2\lambda_2^2 Y \\ Y & -\lambda_2 Y - 1 \end{bmatrix};$$

when $k = 3$, we have that $S = [S_1 \ S_2 \ S_3 \ S_4]$ for

$$\begin{aligned}
 S_1 &= \begin{bmatrix} 5\lambda_2^4\lambda_3^6Y - 30\lambda_2^4\lambda_3^5 - 8\lambda_2^3\lambda_3^7 + 36\lambda_2^3\lambda_3^8 + 3\lambda_2^2\lambda_3^8Y - 6\lambda_2\lambda_3^8 \\ -20\lambda_2^4\lambda_3^5Y + 20\lambda_2^4\lambda_3^4 + 24\lambda_2^3\lambda_3^6Y - 36\lambda_2^2\lambda_3^6 - 4\lambda_2\lambda_3^8Y + 16\lambda_2\lambda_3^7 \\ 30\lambda_2^4\lambda_3^4Y + 20\lambda_2^4\lambda_3^3 - 16\lambda_2^3\lambda_3^5Y - 40\lambda_2^3\lambda_3^4 - 27\lambda_2^2\lambda_3^6Y + 30\lambda_2^2\lambda_3^5 + 12\lambda_2\lambda_3^7Y - 10\lambda_2\lambda_3^6 + \lambda_3^8Y \\ -20\lambda_2^4\lambda_3^3Y - 30\lambda_2^4\lambda_3^2 - 16\lambda_2^3\lambda_3^4Y + 48\lambda_2^2\lambda_3^4Y + 30\lambda_2^2\lambda_3^3 - 8\lambda_2\lambda_3^5Y - 4\lambda_3^7Y \\ 5\lambda_2^4\lambda_3^2Y + 10\lambda_2^4\lambda_3 + 24\lambda_2^3\lambda_3^3Y + 36\lambda_2^3\lambda_3^2 - 27\lambda_2^2\lambda_3^4Y - 36\lambda_2^2\lambda_3^3 - 8\lambda_2\lambda_3^5Y - 10\lambda_2\lambda_3^4 + 6\lambda_3^6Y \\ -8\lambda_2^3\lambda_3^2Y - 16\lambda_2^3\lambda_3 + 12\lambda_2\lambda_3^4Y + 16\lambda_2\lambda_3^3 - 4\lambda_3^5Y \\ 3\lambda_2^2\lambda_3^2Y + 6\lambda_2^2\lambda_3 - 4\lambda_2\lambda_3^3Y - 6\lambda_2\lambda_3^2 + \lambda_3^4Y \end{bmatrix} \\
 S_2 &= \begin{bmatrix} -2\lambda_2^5\lambda_3^6 + 4\lambda_2^4\lambda_3^7 - 2\lambda_2^3\lambda_3^8 \\ -\lambda_2^5\lambda_3^6Y + 6\lambda_2^5\lambda_3^5 + 2\lambda_2^4\lambda_3^7Y - 9\lambda_2^4\lambda_3^6 - \lambda_2^3\lambda_3^8Y + 3\lambda_2^2\lambda_3^8 \\ 4\lambda_2^5\lambda_3^5Y - 4\lambda_2^5\lambda_3^4 - 6\lambda_2^4\lambda_3^6Y + 12\lambda_2^3\lambda_3^6 + 2\lambda_2^2\lambda_3^8Y - 8\lambda_2\lambda_3^7 \\ -6\lambda_2^5\lambda_3^4Y - 4\lambda_2^5\lambda_3^3 + 4\lambda_2^4\lambda_3^5Y + 10\lambda_2^4\lambda_3^4 + 9\lambda_2^3\lambda_3^6Y - 10\lambda_2^3\lambda_3^5 - 6\lambda_2^2\lambda_3^7Y + 5\lambda_2\lambda_3^6 - \lambda_2\lambda_3^8Y - \lambda_3^8 \\ 4\lambda_2^5\lambda_3^3Y + 6\lambda_2^5\lambda_3^2 + 4\lambda_2^4\lambda_3^4Y - 16\lambda_2^3\lambda_3^5Y - 10\lambda_2^3\lambda_3^4 + 4\lambda_2^2\lambda_3^6Y + 4\lambda_2\lambda_3^7Y + 4\lambda_3^7 \\ -\lambda_2^5\lambda_3^2Y - 2\lambda_2^5\lambda_3 - 6\lambda_2^4\lambda_3^3Y - 9\lambda_2^4\lambda_3^2 + 9\lambda_2^3\lambda_3^4Y + 12\lambda_2^2\lambda_3^4 + 4\lambda_2\lambda_3^5Y + 5\lambda_2^2\lambda_3^4 - 6\lambda_2\lambda_3^5Y - 6\lambda_3^6 \\ 2\lambda_2^4\lambda_3^2Y + 4\lambda_2^4\lambda_3 - 6\lambda_2^3\lambda_3^3Y - 8\lambda_2^3\lambda_3^2 + 4\lambda_2\lambda_3^4Y + 4\lambda_3^5 \\ -\lambda_2^3\lambda_3^2Y - 2\lambda_2^3\lambda_3 + 2\lambda_2^2\lambda_3^3Y + 3\lambda_2^2\lambda_3^2 - \lambda_2\lambda_3^4Y - \lambda_3^4 \end{bmatrix} \\
 S_3 &= \begin{bmatrix} 6\lambda_2^8\lambda_3^2 - 16\lambda_2^7\lambda_3^3 + 10\lambda_2^6\lambda_3^4 \\ 3\lambda_2^8\lambda_3^2Y - 6\lambda_2^8\lambda_3 - 8\lambda_2^7\lambda_3^3Y + 5\lambda_2^6\lambda_3^4Y + 36\lambda_2^6\lambda_3^3 - 30\lambda_2^5\lambda_3^4 \\ -4\lambda_2^8\lambda_3^2Y + 16\lambda_2^7\lambda_3 + 24\lambda_2^6\lambda_3^3Y - 36\lambda_2^6\lambda_3^2 - 20\lambda_2^5\lambda_3^4Y + 20\lambda_2^4\lambda_3^4 \\ \lambda_2^8Y + 12\lambda_2^7\lambda_3^3Y - 27\lambda_2^6\lambda_3^2Y - 10\lambda_2^6\lambda_3 - 16\lambda_2^5\lambda_3^3Y + 30\lambda_2^5\lambda_3^2 + 30\lambda_2^4\lambda_3^4Y - 40\lambda_2^4\lambda_3^3 + 20\lambda_2^3\lambda_3^4 \\ -4\lambda_2^7Y - 8\lambda_2^6\lambda_3Y + 48\lambda_2^5\lambda_3^2Y - 16\lambda_2^5\lambda_3^3Y + 30\lambda_2^4\lambda_3^3 - 20\lambda_2^3\lambda_3^4Y - 30\lambda_2^2\lambda_3^4 \\ 6\lambda_2^6Y - 8\lambda_2^5\lambda_3Y - 27\lambda_2^4\lambda_3^2Y - 10\lambda_2^4\lambda_3 + 24\lambda_2^3\lambda_3^3Y - 36\lambda_2^3\lambda_3^2 + 5\lambda_2^2\lambda_3^4Y + 36\lambda_2^2\lambda_3^3 + 10\lambda_2\lambda_3^4 \\ -4\lambda_2^5Y + 12\lambda_2^4\lambda_3Y + 16\lambda_2^3\lambda_3 - 8\lambda_2^2\lambda_3^3Y - 16\lambda_2\lambda_3^3 \\ \lambda_2^4Y - 4\lambda_2^3\lambda_3Y + 3\lambda_2^2\lambda_3^3Y - 6\lambda_2\lambda_3^3 + 6\lambda_2\lambda_3^2 \end{bmatrix} \\
 S_4 &= \begin{bmatrix} -2\lambda_2^8\lambda_3^3 + 4\lambda_2^7\lambda_3^4 - 2\lambda_2^6\lambda_3^5 \\ -\lambda_2^8\lambda_3^3Y + 3\lambda_2^8\lambda_3^2 + 2\lambda_2^7\lambda_3^4Y - \lambda_2^6\lambda_3^5Y - 9\lambda_2^6\lambda_3^4 + 6\lambda_2^5\lambda_3^5 \\ 2\lambda_2^8\lambda_3^2Y - 8\lambda_2^7\lambda_3^3 - 6\lambda_2^6\lambda_3^4Y + 12\lambda_2^5\lambda_3^4 + 4\lambda_2^5\lambda_3^5Y - 4\lambda_2^4\lambda_3^5 \\ -\lambda_2^8\lambda_3^2Y - \lambda_2^8 - 6\lambda_2^7\lambda_3^3Y + 9\lambda_2^6\lambda_3^4Y + 5\lambda_2^5\lambda_3^4Y + 4\lambda_2^5\lambda_3^5Y - 10\lambda_2^4\lambda_3^4 - 6\lambda_2^4\lambda_3^5Y + 10\lambda_2^4\lambda_3^4 - 4\lambda_2^3\lambda_3^5 \\ 4\lambda_2^7\lambda_3^3Y + 4\lambda_2^7 + 4\lambda_2^6\lambda_3^3Y - 16\lambda_2^5\lambda_3^3Y + 4\lambda_2^4\lambda_3^4Y - 10\lambda_2^4\lambda_3^3 + 4\lambda_2^3\lambda_3^5Y + 6\lambda_2^2\lambda_3^5 \\ -6\lambda_2^6\lambda_3^3Y - 6\lambda_2^6 + 4\lambda_2^5\lambda_3^3Y + 9\lambda_2^4\lambda_3^3Y + 5\lambda_2^4\lambda_3^2 - 6\lambda_2^3\lambda_3^4Y + 12\lambda_2^3\lambda_3^3 - \lambda_2^2\lambda_3^5Y - 9\lambda_2^2\lambda_3^4 - 2\lambda_2\lambda_3^5 \\ 4\lambda_2^5\lambda_3^3Y + 4\lambda_2^5 - 6\lambda_2^4\lambda_3^3Y - 8\lambda_2^3\lambda_3^3 + 2\lambda_2^2\lambda_3^4Y + 4\lambda_2\lambda_3^4 \\ -\lambda_2^4\lambda_3^3Y - \lambda_2^4 + 2\lambda_2^3\lambda_3^3Y - \lambda_2^2\lambda_3^3Y + 3\lambda_2^2\lambda_3^2 - 2\lambda_2\lambda_3^3 \end{bmatrix}
 \end{aligned}$$

The polynomial $\nu(\lambda)$ is a constant zero polynomial (i.e., $\nu_1 = \dots = \nu_{4k-4} = 0$) if and only if the vector $[\Delta_3 \ \Delta_4 \ \dots \ \Delta_{2k}]^\top$ lies in the zero space of the matrix S . Let L be the matrix formed by the first $2k - 2$ rows of S . For $2 \leq k \leq 5$, our calculation shows that the determinant of L is equal to

$$|L| = (2k - 1)! \prod_{j \in T} \lambda_j^{8k-12} \prod_{j_1, j_2 \in T; j_1 \neq j_2} (\lambda_{j_1} - \lambda_{j_2})^{8k-12}.$$

Since $\{\lambda_j\}_{j \in [k]}$ are distinct and chosen from \mathbb{F}_p^* , the determinant $|L|$ is always nonzero. Therefore the matrix S is of full column rank for $2 \leq k \leq 5$, which indicates that the zero space of the matrix S is $\{0\}$. It follows that $[\Delta_3 \ \Delta_4 \ \dots \ \Delta_{2k}]^\top = 0$, which indicates that the polynomial $\hat{f}(\lambda)$ and $f(\lambda)$ are interpolated with the same $2k$ values. Hence, $\hat{f}_0 = f_0$ and Eq. (10a) does not hold anymore. There are

$$\begin{aligned}
 \Delta = 0 &\Leftrightarrow [\Delta_3 \ \Delta_4 \ \dots \ \Delta_{2k}]^\top = 0 \Rightarrow \neg \text{Eq. (10a)}, \\
 \text{Eq. (10a)} &\Rightarrow [\Delta_3 \ \Delta_4 \ \dots \ \Delta_{2k}]^\top \neq 0 \Leftrightarrow \Delta \neq 0.
 \end{aligned} \tag{19}$$

Therefore, it is impossible for Adv to win by setting $\nu(\lambda)$ to be a constant zero polynomial for $2 \leq k \leq 5$. Therefore, $\nu(\lambda)$ is a nonzero polynomial and λ_1 is uniformly distributed in $\mathbb{F}_p \setminus \{\lambda_j\}_{j \in T}$ in Adv's point of view for $2 \leq k \leq 5$. Since

$\nu(\lambda)$ is of degree $4k - 4$, it has at most $4k - 4$ zeros.

$$\Pr \left[\nu(\lambda_1) = 0 \mid \Delta \neq 0 \right] \leq \frac{4k - 4}{p - k}.$$

By Eqs. (11), (13), (16) and (19), the adversary Adv wins the experiment with probability

$$\begin{aligned} \Pr \left[\text{EXP}_{\text{Adv}, \Gamma}^{\text{Ver}}(n, DB, i, T) = 1 \right] &= \Pr \left[\text{Eq. (10a)} \wedge \text{Eq. (10b)} \right] \\ &\leq \Pr \left[\Delta \neq 0 \wedge |N| = 0 \right] \\ &\leq \Pr \left[|N| = 0 \mid \Delta \neq 0 \right] \\ &\leq \frac{4k - 4}{p - k}. \end{aligned}$$

□

For any statistical security parameter κ , we can choose the size p of the finite field \mathbb{F}_p as $p = O(2^{\kappa} k)$. Then our k -server PIR-RV protocol Γ is ϵ -secure with $\epsilon = (4k - 4)/(p - k)$, which is negligible in κ . In practice, κ may be chosen based on the requirement of security level. For example, if 128-bit security is required, then we may choose $\kappa \approx 128$.

Efficiency Analysis. Clearly, the client sends a length- m vector in \mathbb{F}_p to each server and each server returns a length- $(m + 1)$ vector in \mathbb{F}_p . The communication complexity of our t -private k -server PIR-RV protocol is $O(km \log p)$. Note that m is an integer such that there is a binary $(m, 4, w)$ -constant weight code C of size $\geq n$, the size of the database DB . By Theorem 1, we can choose m such that $\binom{m}{w}/m \geq n$. Hence, $m = O(w(nw)^{1/(w-1)}) = O\left(\frac{k}{t} \left(\frac{nk}{t}\right)^{1/(\lfloor (2k-1)/t \rfloor - 1)}\right)$, the communication complexity of Γ is $\text{CC}_{\Gamma}(n, k) = O\left(\frac{k^2}{t} \left(\frac{nk}{t}\right)^{1/(\lfloor (2k-1)/t \rfloor - 1)} \log p\right)$. In PIR-RV/PIR, the parameters k and t are relatively small. For any given k and t , we mostly care about the growth rate of the communication complexity of a PIR-RV/PIR protocol in the parameter n , the size of the database. When $k = 2$ and $t = 1$, the communication complexity of Γ is $O(n^{1/2} \log p)$, which is consistent with the 2-server PIR-RV protocol in [13].

The prime p in our PIR-RV protocol should be much larger than the p in Woodruff-Yekhanin PIR [18] due to the requirement of security. The larger p may incur higher communication complexity. However, this side effect can be removed by using the protocol to retrieve elements from a database in \mathbb{F}_p^n such that every time $\log p$ bits are retrieved but no extra communication is needed. In fact, if we use the protocol in such a way, then the average communication cost incurred by retrieving one bit from the database can be reduced to $O\left(\frac{k^2}{t} \left(\frac{nk}{t}\right)^{1/(\lfloor (2k-1)/t \rfloor - 1)}\right)$, which is independent of p .

5 Conclusion

In this paper, we define a k -server PIR-RV model for any $k \geq 2$ and propose a construction of t -private k -server PIR-RV protocols out of Woodruff and

Yekhanin's t -private k -server PIR protocols, by extending the idea of [13]. Proving the security of our protocols is highly non-trivial. We give a novel technique to prove the security of the k -server PIR-RV protocols for $2 \leq k \leq 5$. It is an interesting open problem to extend our proof for arbitrary k .

Acknowledgement. The authors thank the anonymous reviewers for their helpful comments. This work was supported by Natural Science Foundation of Shanghai (No. 21ZR1443000) and National Natural Science Foundation of China (No. 61602304).

References

1. Angel, S., Setty, S.: Unobservable communication over fully untrusted infrastructure. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 551–569 (2016)
2. Beimel, A., Ishai, Y.: Information-theoretic private information retrieval: a unified construction. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 912–926. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-48224-5_74
3. Beimel, A., Ishai, Y., Kushilevitz, E.: General constructions for information-theoretic private information retrieval. *J. Comput. Syst. Sci.* **71**(2), 213–247 (2005)
4. Beimel, A., Ishai, Y., Kushilevitz, E., Raymond, J.F.: Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In: The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings, pp. 261–270. IEEE (2002)
5. Beimel, A., Stahl, Y.: Robust information-theoretic private information retrieval. In: Cimato, S., Persiano, G., Galdi, C. (eds.) SCN 2002. LNCS, vol. 2576, pp. 326–341. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36413-7_24
6. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: 1995 Proceedings, 36th Annual Symposium on Foundations of Computer Science (1995)
7. Devet, C., Goldberg, I., Heninger, N.: Optimally robust private information retrieval. *Cryptology ePrint Archive* (2012)
8. Dvir, Z., Gopi, S.: 2-server PIR with subpolynomial communication. *J. ACM (JACM)* **63**(4), 1–15 (2016)
9. Efremenko, K.: 3-query locally decodable codes of subexponential length. In: Proceedings of the Forty-first Annual ACM Symposium on Theory Of Computing, pp. 39–44 (2009)
10. Goldberg, I.: Improving the robustness of private information retrieval. In: 2007 IEEE Symposium on Security and Privacy (SP2007), pp. 131–148. IEEE (2007)
11. Graham, R., Sloane, N.: Lower bounds for constant weight codes. *IEEE Trans. Inf. Theory* **26**(1), 37–43 (1980)
12. Gupta, T., Crooks, N., Mulhern, W., Setty, S., Walfish, M.: Scalable and private media consumption with popcorn. In: *Unix Symposium on Networked Systems Design & Implementation* (2015)
13. Ke, P., Zhang, L.F.: Two-server private information retrieval with result verification. In: 2022 IEEE International Symposium on Information Theory (ISIT), pp. 408–413. IEEE (2022)

14. Khoshgozaran, A., Shahabi, C.: Private information retrieval techniques for enabling location privacy in location-based services. In: Bettini, C., Jajodia, S., Samarati, P., Wang, X.S. (eds.) *Privacy in Location-Based Applications*. LNCS, vol. 5599, pp. 59–83. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03511-1_3
15. Kurosawa, K.: How to correct errors in multi-server PIR. In: Galbraith, S.D., Moriai, S. (eds.) *ASIACRYPT 2019*. LNCS, vol. 11922, pp. 564–574. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34621-8_20
16. Poole, D.: *Linear algebra: a modern introduction*. Cengage Learning (2014)
17. Shamir, A.: How to share a secret. *Commun. ACM* **22**(11), 612–613 (1979)
18. Woodruff, D., Yekhanin, S.: A geometric approach to information-theoretic private information retrieval. In: *20th Annual IEEE Conference on Computational Complexity (CCC2005)*, pp. 275–284. IEEE (2005)
19. Yang, E.Y., Xu, J., Bennett, K.H.: Private information retrieval in the presence of malicious failures. In: *Proceedings 26th Annual International Computer Software and Applications*, pp. 805–810. IEEE (2002)
20. Yannuzzi, M., Milito, R., Serral-Gracià, R., Montero, D., Nemirovsky, M.: Key ingredients in an IoT recipe: Fog computing, cloud computing, and more fog computing. In: *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 325–329. IEEE (2014)
21. Yekhanin, S.: Towards 3-query locally decodable codes of subexponential length. *J. ACM (JACM)* **55**(1), 1–16 (2008)
22. Zhang, L.F., Safavi-Naini, R.: Verifiable multi-server private information retrieval. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) *ACNS 2014*. LNCS, vol. 8479, pp. 62–79. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07536-5_5
23. Zhang, L.F., Wang, H., Wang, L.P.: Byzantine-robust private information retrieval with low communication and efficient decoding. In: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pp. 1079–1085 (2022)
24. Zhao, L., Wang, X., Huang, X.: Verifiable single-server private information retrieval from LWE with binary errors. *Inf. Sci.* **546**, 897–923 (2021)
25. Zhu, L., Lin, C., Lin, F., Zhang, L.F.: Post-quantum cheating detectable private information retrieval. In: Meng, W., Jensen, C.D. (eds.) *ICT Systems Security and Privacy Protection. SEC 2022. IFIP Advances in Information and Communication Technology*, vol. 648, pp. 431–448. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-06975-8_25

Isogeny-Based Cryptography



Practical Robust DKG Protocols for CSIDH

Shahla Atapoor¹, Karim Baghery¹, Daniele Cozzo^{1,2(✉)}, and Robi Pedersen¹

¹ imec-COSIC, KU Leuven, Leuven, Belgium
{shahla.atapoor,karim.baghery,robi.pedersen}@kuleuven.be
² IMDEA Software Institute, Madrid, Spain
daniele.cozzo@imdea.org

Abstract. A Distributed Key Generation (DKG) protocol is an essential component of threshold cryptography. DKGs enable a group of parties to generate a secret and public key pair in a distributed manner so that the secret key is protected from being exposed, even if a certain number of parties are compromised. Robustness further guarantees that the construction of the key pair is always successful, even if malicious parties try to sabotage the computation. In this paper, we construct two efficient robust DKG protocols in the CSIDH setting that work with Shamir secret sharing. Both the proposed protocols are proven to be actively secure in the quantum random oracle model and use an Information Theoretically (IT) secure Verifiable Secret Sharing (VSS) scheme that is built using bivariate polynomials. As a tool, we construct a new piecewise verifiable proof system for structured public keys, that could be of independent interest. In terms of isogeny computations, our protocols outperform the previously proposed DKG protocols CSI-RAShI and Structured CSI-RAShI. As an instance, using our DKG protocols, 4 parties can sample a PK of size 4 kB, for CSI-FiSh and CSI-SharK, respectively, 3.4 and 1.7 times faster than the current alternatives. On the other hand, since we use an IT-secure VSS, the communication cost of our schemes is generally higher, except for a few specific parameters, and the fraction of corrupted parties is limited to less than a third.

Keywords: Distributed Key Generation · CSIDH · Isogenies · VSS

1 Introduction

Key management is an important aspect of cryptography, as the security of cryptographic algorithms crucially relies on the safety of secret keys. If an attacker obtains the secret key, the security of the system is generally compromised. Threshold cryptography addresses this issue by dividing the secret key among a group of devices, in such a way that only subsets of these devices above a specific threshold size t can reconstruct the secret key. An attacker would then need to obtain shares from $t + 1$ parties to reconstruct the secret key. This effectively eliminates the single point of failure of cryptographic algorithms. At the basis

of threshold schemes are Distributed Key Generation (DKG) protocols, that are run by the parties to produce a correct sharing of the secret key along with the associated public key.

Threshold protocols based on classical assumptions have been extensively studied and an initiative to standardize them has been taken by NIST.¹ However, due to the vulnerability of classical assumptions, like the Discrete Logarithm (DL) and factoring problems, against quantum computers [28], the cryptographic community has begun to work on a general migration of public key cryptography to algorithms that are based on post-quantum problems. Threshold cryptography will also need to adapt to these new problems.

One active research direction for designing post-quantum secure protocols is isogeny-based cryptography, which relies on the hard problem of finding a secret isogeny between two public elliptic curves. Isogeny-based cryptography was initially based on ordinary elliptic curves [11, 12, 26], but in the last decade, focus has almost entirely switched to supersingular elliptic curves due to security and/or efficiency reasons. One of the first protocols to propose using supersingular elliptic curves was SIDH (supersingular isogeny-based Diffie-Hellman) [15]. However, protocols based on SIDH need parties to reveal extra information along with their public key, which recently has been proven to be enough to recover the secret isogeny [9, 22, 25], thus effectively breaking the SIDH-based family of protocols. Fortunately, newer schemes such as SQISign [16] and CSIDH [10] (commutative SIDH) are unaffected by these attacks, as they do not need to reveal this extra information. The intrinsic commutativity of CSIDH has shown to provide a lot of flexibility when designing cryptographic protocols. As such, CSIDH provides a versatile toolbox, which allows to build more complex cryptographic protocols, such as threshold schemes [2, 6, 8, 13, 17].

Previous DKGs in the CSIDH Setting. The first CSIDH-based actively secure DKG was proposed as part of the distributed signature scheme called Sashimi [13]. However, their original DKG protocol is quite inefficient in practice. In follow-up work, Beullens, Disson, Pedersen and Vercauteren [6] proposed the first *robust* DKG protocol for CSIDH, which works with Shamir secret sharing and allows a set of parties to sample a single public key in a fully distributed manner. The robustness ensures that the parties are able to carry on the DKG even if the adversary tries to sabotage the protocol. Their construction, dubbed CSI-RAShI, consists of two phases. A Verifiable Secret Sharing (VSS) phase, where the parties create their shares of the secret key, and the Public Key (PK) computation phase, where the parties use their shares to compute the target PK. While in traditional Pedersen VSS [24], the verification of shares was easily done by exploiting the homomorphic properties of modular exponentiation, this is not possible with elliptic curves. To deal with this concern, [6] introduces Piecewise Verifiable Proofs (PVPs), a sort of Zero-Knowledge (ZK) proof that allows parties to verify that the share they got is consistent with respect to some public commitment. In the PK computation step, a different ZK proof is used. Both these proofs use a binary challenge space and as a result, the protocol is still

¹ See <https://csrc.nist.gov/projects/threshold-cryptography>.

computationally expensive, albeit faster than the Sashimi proposal. Moreover, as a result of the ZK proofs used in the PK computation step, the security of the DKG protocol needs to rely on a decisional assumption. As a result, the authors can only argue security when the target public key is uniformly random, rather than arbitrary.

CSI-RAShI is constructed to sample a single public key $[x]E_0$ in a distributed manner, and by repeating it k times, one can sample extended public keys of the form $([x_1]E_0, \dots, [x_k]E_0)$ as they are used in the signature scheme CSI-FiSh [7]. Repeating CSI-RAShI k times is very inefficient, even if optimized [2]. To deal with that concern, the authors of [2] proposed CSI-SharK, as a variant of CSI-FiSh with Sharing-friendly Key, as well as Structured CSI-RAShI, as a variant of CSI-RAShI for Structured Public Keys (SPKs). Structured CSI-RAShI allows a set of parties to sample an SPK (introduced in [3]), i.e. a public key of the type $([c_1x]E_0, \dots, [c_kx]E_0)$, where c_1, \dots, c_k define an exceptional set. For an SPK of the same size k , Structured CSI-RAShI is 4 times faster than generating an extended public key using the original CSI-RAShI.

Our Contribution. In this paper, we construct two efficient robust DKG protocols for CSIDH-based cryptographic primitives that work with Shamir secret sharing. Both protocols are proven to be actively secure in the Quantum Random Oracle Model (QROM). Our DKG protocols can be considered as an alternative to the DKG protocols CSI-RAShI [6] and its Structured variant [2]. We show that both our proposed DKG protocols outperform these previous proposals in terms of computational cost. Moreover, the VSS step of our DKG protocols does not rely on any decisional (or computational) assumption, does not use ZK proofs, and instead uses an efficient Information Theoretically (IT) secure VSS scheme. However, the latter comes at the cost of increased communication and a higher number of needed honest parties, in comparison with the alternatives. Specifically, in the VSS step, more than $2/3$ of the parties must be honest rather than just the majority of them. But, as we are in the static corruption setting, for the PK computation phase still our protocols work in the majority honest setting, as in the original CSI-RAShI and its structured variant.

Technical Overview. To design our proposed DKG protocols, we have modified the construction of CSI-RAShI to minimize the number of isogeny computations required for each party, as they are the most computationally expensive parts in these protocols. Our first key modification is to change the secret sharing step of CSI-RAShI to an efficient VSS scheme based on bivariate polynomials, that was first proposed in [31], but for a different purpose. The idea is that each party P_i can distribute a value $x^{(i)}$ by sampling a random polynomial $q^{(i)}(Z)$ of degree t subject to $q^{(i)}(0) = x^{(i)}$, then hiding $q^{(i)}(Z)$ in the bivariate polynomial $S^{(i)}(X, Y)$, also of degree t in both variables, in a way that $S^{(i)}(0, Z) = q^{(i)}(Z)$. Now, instead of sending evaluations of a univariate polynomial, the verification shares are two polynomials defined by $S^{(i)}(j, Y) = g_j(Y)$ and $S^{(i)}(X, j) = f_j(X)$, which are then sent to each other party P_j . Parties can now do pairwise checks to verify that the polynomials they got from P_i are correct by testing whether

$f_j^{(i)}(k) = g_k^{(i)}(j)$ and $f_k^{(i)}(j) = g_j^{(i)}(k)$. The key point is that if there are at least t of these relations that are satisfied, then there exists a unique bivariate polynomial $S^{(i)}(X, Y)$ as above and P_i acted honestly.

After the VSS step, to compute the PK, parties use their secret shares and engage in a round-robin MPC protocol and prove that they did their computation correctly. As opposed to CSI-RAShi, parties can no longer use the ZK proof from [13] to prove correct execution, since they did not commit to their shares $x^{(i)}$ in the VSS. Instead, we observe that as a result of our modifications, we can use PVPs for this purpose. With PVPs, parties are able to prove that they are updating the PK using $q^{(i)}(0) = x^{(i)}$, which was shared in the VSS step.

Using the bivariate polynomial-based VSS and our new modifications improves the efficiency of the final DKG protocol, when compared to previous results in the literature. The first source of improvement is that, instead of performing expensive PVPs and isogeny computations to verify the shares, as done in the VSS step of CSI-RAShi, now the parties just need to perform fast polynomial evaluations and pairwise comparisons. The second source of improvement is that we replace the ZK proofs in the PK computation step with a PVP scheme, which is more efficient. Moreover, we remove the need for a decisional assumption and also achieve IT security in the VSS step. Conveniently, removing the need for a decisional assumption, allows us to use the twist technique from [7], to extend the challenge space of the PVP scheme, and further improve the communication and computational costs in some cases. A downside resulting from our modifications is that the communication generally increases (with a few exceptions), and the fraction of corrupted parties allowed in the VSS reduces to less than a third, rather than half, which seems to be unavoidable for IT security. We also show how to build extended public keys with this protocol and discuss optimizations to reduce the overall runtime.

The second DKG protocol we propose allows a set of parties to sample an SPK. Note that since SPKs only have a single secret key, the VSS step is independent of the length of the SPK. In order to adapt the PK computation step to SPKs, we also construct a new PVP scheme, which is specific to SPKs, called Structured PVPs (SPVP), which we believe to be of independent interest. The SPVP scheme allows parties to prove that they are computing/updating the target SPK using different factors of the $x^{(i)}$ shared in the VSS step. By replacing the ZK proofs in the SPK computation with our new SPVPs, we also manage to improve the efficiency of the Structured CSI-RAShi scheme, proposed in [2].

At the end, it is worth mentioning that the idea of using an IT secure VSS based on bivariate polynomials within a DKG was already exploited by [35], in the DL setting. In that work, the authors plug the VSS scheme of [31] into the threshold scheme by Gennaro, Jarecki, Krawczyk and Rabin [20] for distributing the key generation of the Schnorr signature. After the VSS, the PK is then constructed by having the players aggregate all partial public keys as specified in [20]. However in the DL setting using the bivariate polynomial-based VSS [31] does not provide a big gain in terms of efficiency, in comparison with [20]. On the other hand, for isogenies, as we show the advantage of using a bivariate

polynomial based VSS scheme in CSI-RAShi [6] and Structured CSI-RAShi [2] is significant. As we can save on expensive ZK proofs and on the number of isogeny computations (each taking around 35–40 ms).

Efficiency. Our base protocol (for a single public key) is naturally about twice as fast as CSI-RAShi, currently the state-of-the-art. On the downside, communication cost scales quadratically in the number of parties n , instead of linearly, which leads to a noticeable increase when more parties are present but manageable for low n . A noticeable exception is $n = 2$, where the communication cost of our scheme is actually 27% lower than CSI-RAShi's and the computational gain is almost 3. These extra results are mainly due to the higher impact of using the twist trick with fewer parties.

For larger public keys, we can use optimizations similar to the ones proposed in [2]. The gains depend on the number of parties n and public key sizes k , but we show that in terms of the number of isogeny computations, our protocols always outperform the results from [2]. Due to the twist trick, the most important gains are visible for low n . But even for $n \rightarrow \infty$, we outperform extended CSI-RAShi by a factor of 3 and structured CSI-RAShi with approximately a factor $1+k^{-1/2}$. The gain against the latter thus becomes negligible only for large k and large n .

As a numerical example, consider PKs with $k = 2^6$ elements (which have a size of about 4kB). For $n = 4$, we get a gain of 3.4 and 1.7 against extended and structured CSI-RAShi from [2], while for $n = 40$, this gain reduces to 3.0 and 1.5, respectively. Comparing the communications of our structured scheme and structured CSI-RAShi for $k = 2^6$, our scheme has about 8% higher communication when $n = 4$, while for $n = 40$, we have a factor of 60 more communication.

Outline. In Sect. 2, we review some preliminary concepts. In Sect. 3, we first discuss the VSS based on bivariate polynomials, then use it to construct a DKG protocol for a PK with a single curve and finally discuss its extension for generating multiple independent curves. Then, in Sect. 4, we present the second DKG protocol for sampling an SPK. We discuss and compare the efficiency of our DKG protocols in Sect. 5. Finally, we conclude the paper in Sect. 6.

2 Preliminaries

Notation. We use the assignment operator \leftarrow to denote random sampling from a probability distribution D , e.g. $x \leftarrow D$, or uniform sampling from a set X , e.g. $x \leftarrow X$. We write λ to denote a security parameter. We call a function f *negligible in X* , if for any constant c , there exists some X_0 , such that $f(X) < X^{-c}$ for $X > X_0$. We denote this as $\text{negl}(X)$. We call a function simply *negligible* if it is negligible in λ . We write $\mathbb{Z}_N := \mathbb{Z}/N\mathbb{Z}$ and $\log(x) := \log_2(x)$.

2.1 Isogeny-Based Cryptography

Isogenies are rational maps between elliptic curves, that are also surjective homomorphisms with respect to the natural group structure between these curves. In

this work, we will only consider supersingular elliptic curves and separable isogenies defined over prime fields \mathbb{F}_p . We denote the set of such elliptic curves as \mathcal{E} . The endomorphisms of elliptic curves over \mathbb{F}_p define a ring structure that is isomorphic to orders \mathcal{O} in the quadratic imaginary field $\mathbb{Q}(\sqrt{-p})$. Separable isogenies are uniquely defined by their kernels, which in turn can be identified with the kernels of ideal classes in the class group $\text{cl}(\mathcal{O})$. For efficiency reasons, the prime p is chosen, so that $p - 1 = 4 \prod_i \ell_i$ consists of the multiplication of small primes. By this choice, ideals of the type $\ell_i \mathcal{O}$ split into a prime ideal \mathfrak{l}_i and its conjugate $\bar{\mathfrak{l}}_i$, uniquely defining an isogeny and its dual, both of small prime (and thus efficiently computable) prime degree ℓ_i .² Throughout this work, we assume the class group for the relevant order $\text{cl}(\mathcal{O})$ to be known, so that arbitrary ideals can be transformed into efficient isogeny computations using the relation lattice, by translating them to a small number of consecutive degree- ℓ_i isogeny computations.³

We note that in general, class groups are of composite order. For any cyclic subgroup of the class group, of size $N \mid \#\text{cl}(\mathcal{O})$ with generator \mathfrak{g} , we can then define isogenies through the action of elements in $\mathbb{Z}_N \subseteq \mathbb{Z}_{\#\text{cl}(\mathcal{O})}$ as $[\] : \mathbb{Z}_N \times \mathcal{E} \rightarrow \mathcal{E}$, where the action $[a]E \mapsto E'$ defines an isogeny with kernel isomorphic to $\ker \mathfrak{g}^a$, reduced modulo the relation lattice. In this way, the map $[\]$ defines a free and transitive group action [12] by \mathbb{Z}_N (as a proxy for the subgroup of $\text{cl}(\mathcal{O})$) on the set \mathcal{E} . We refer the reader to [5, 7, 10, 34] for more details on the explicit computations of isogenies. For a more thorough introduction to isogenies and isogeny-based cryptography, the authors recommend [10, 14, 30].

The fact that N can be composite implies that in general \mathbb{Z}_N constitutes a ring (and not a field). This has to be handled with care, as in some applications, inverse elements are needed. Assuming that N has the prime decomposition $\prod_{i=1}^n N_i$, where, for later reference, we assume $N_1 < \dots < N_n$, we can easily work in a subgroup of size $N' = \prod_{i \in S} N_i$ for $S \subset \{1, \dots, n\}$ by using the generator $\mathfrak{g}^{N/N'}$.

2.2 DKG Protocols and Piecewise Verifiable Proofs

A DKG protocol mainly consists of secret sharing and PK computation. Secret sharing-based protocols are interactive schemes between n parties P_1, \dots, P_n , such that at the end of the protocol, each party holds a share of a common secret s in a way, that only specifically allowed sets (qualified sets) can join

² The extra factor 4 is chosen so that $p \equiv 3 \pmod{4}$, which makes the particular curve $E_0 : y^2 = x^3 + x$ supersingular, and allows to work in the more efficient Montgomery coordinates, see [10] for more details.

³ We note that this is not a trivial assumption, since computing large class groups is generally difficult using classical computers, cf. [7], which computed a 257-bit class group and associated lattice of relations for the CSIDH-512 parameter set from [10]. An alternative approach is discussed in [18], which strongly speeds up the class group computations, but unfortunately leads to much slower group action computations. We note however that there exist efficient quantum algorithms [21] for this purpose.

forces in order to efficiently reconstruct s (or equivalently act with the isogeny $[s]$), while this is unfeasible for any non-qualified set.

Throughout this work, we realize this sharing using Shamir Secret Sharing (SSS) [27] and variants thereof. In SSS, the secret is defined as the evaluation of some secret polynomial $q(X) \in \mathbb{Z}_N[X]$ at zero, i.e. $s = q(0) \in \mathbb{Z}_N$. The shares that the parties hold are then just evaluations of $q(X)$ at other positions than at (typically) zero. For simplicity we fix the share of party P_i to be $s_i = q(i)$ for $i \in \{1, \dots, n\}$. The degree of the secret polynomial, $t = \deg q$, is then the determining factor of qualified sets, i.e. any set Q of size $|Q| > t$ can use Lagrange interpolation in order to reconstruct the secret as follows

$$s = q(0) = \sum_{i \in Q} q(i)L_i^Q = \sum_{i \in Q} q(i) \prod_{j \in Q \setminus \{i\}} \frac{j}{j-i} \pmod N,$$

where L_i^Q are the Lagrange coefficients for the set Q . For a set of size $\leq t$, the value s cannot be reconstructed, and in fact is information-theoretically hidden. We note that since N might be a composite number in our case, we have to ensure that $n < N_1$ [17], where N_1 is the smallest (non-trivial) divisor of N so that any difference $j - i$ is guaranteed to be invertible mod N . In case we want to allow more than N_1 parties, we can work in the subgroup generated by \mathfrak{g}^{N_1} , as explained in Sect. 2.1. Throughout the rest of this work, for simplicity, we always assume N to be the size of the subgroup that we are working with, i.e. the largest subgroup $\mathbb{Z}_N \subseteq \mathbb{Z}_{\#\text{cl}(\mathcal{O})}$, for which $N_1 > n$, where N_1 is the smallest non-trivial divisor of N .

In the easiest case, the shares s_i are produced by a trusted third party, called a dealer. However, a Verifiable Secret Sharing (VSS) can easily be turned into a DKG protocol, where the secret polynomial is generated by the parties themselves. A direct way to achieve this is outlined in [23], where each party P_i takes the role of a dealer and generates a polynomial $q^{(i)}(X) \in \mathbb{Z}_N[X]_t$ of the correct degree and privately sends the share $q^{(i)}(j)$ to the party P_j for $j \in \{1, \dots, n\} \setminus \{i\}$. Then, every party can locally compute their own share by summing all the shares. This implicitly defines the polynomial $q(X) = \sum_{i=1}^n q^{(i)}(X)$ and each player's share $q(j) = \sum_{i=1}^n q^{(i)}(j)$ as the sum of their shares. The implicitly defined secret is $s = q(0)$, unknown to all players, assuming the honest-majority setting and that the protocol has been executed properly.

Since some (up to t) parties might behave maliciously, parties need a way to verify that the shares they get are correct, i.e. that these really are the shares of a polynomial of degree at most t . It is clear, that if a malicious adversary chooses a polynomial of a degree larger than t , the reconstruction will fail. We therefore define the functionality of a Shamir-based VSS as follows.

Definition 2.1 ([1, Functionality 5.5]). *We define the VSS-functionality*

$$F_{VSS}(q(X), n) = \begin{cases} q(1), \dots, q(n) & , \text{ if } \deg q \leq t, \\ \perp, \dots, \perp & , \text{ otherwise,} \end{cases}$$

taking as input a $q(X) \in \mathbb{Z}_N[X]_t$ of degree t , and each party $i \in \{1, \dots, n\}$ receiving a Shamir share $q(i)$. If $\deg q > t$, the parties output \perp instead.

As mentioned before, the next step in a DKG is the PK computation. In isogeny-based protocols, this means generating one or several elliptic curves, e.g. $E = [s]E_0$, by jointly computing the action $[s]$ in a distributed manner. Since different elliptic curves cannot be combined, the distributed computation of $[s]$ has to be done in a round-robin fashion [17].

Security Requirements of DKG Protocols. Next, we recall the security requirements of DKG protocols based on SSS. We stick to the definitions outlined in [19], using the notation introduced in [6].⁴ We denote by $A, B \leftarrow \langle \mathcal{A}^{\mathcal{O}_1}(X) \mid \mathcal{B}^{\mathcal{O}_2}(Y) \rangle$ the joint execution of the DKG protocol by (sets of) parties \mathcal{A} with input X and oracle access to \mathcal{O}_1 and \mathcal{B} with input Y and oracle access to \mathcal{O}_2 , and yielding the output distributions A for \mathcal{A} and B for \mathcal{B} , respectively. We define the output distributions of the DKG protocol as follows

$$D_{\text{out}}(\mathcal{A}^{\mathcal{O}_1}(X), \mathcal{B}^{\mathcal{O}_2}(Y)) = \{(A, B) \mid A, B \leftarrow \langle \mathcal{A}^{\mathcal{O}_1}(X) \mid \mathcal{B}^{\mathcal{O}_2}(Y) \rangle\},$$

and drop the inputs, whenever the input string is empty.

Definition 2.2 (Robust correctness). *A DKG protocol based on SSS between n parties P_1, \dots, P_n is called correct, if for any PPT adversary \mathcal{A} and any subset $I \subseteq \{1, \dots, n\}$ of size $|I| > t$ and $n - |I| \leq t$ for any $t < n$, we have that*

$$\Pr \left[\begin{array}{l} \nexists f \in \mathbb{Z}_N[x]_t : \\ E_1 = \dots = E_n = [f(0)]E_0, \\ \text{and } \forall i \in I : f(i) = s_i \end{array} \middle| A, \{(E_i, s_i)\}_{i \in I} \leftarrow \langle \mathcal{A}^{\mathcal{O}} \mid \{P_i^{\mathcal{O}}\}_{i \in I} \rangle \right] \leq \text{negl}(\lambda).$$

Definition 2.3 (Secrecy). *Let \mathcal{O} be a random oracle. A DKG protocol based on SSS between n parties P_1, \dots, P_n satisfies secrecy, if for any PPT adversary \mathcal{A} , and any subset $I \subseteq \{1, \dots, n\}$ with $|I| > t$ and $n - |I| \leq t$, there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that the following distributions are indistinguishable⁵*

$$D_{\text{out}}(\mathcal{A}^{\mathcal{O}} \mid \{P_i^{\mathcal{O}}\}_{i \in I}) \approx D_{\text{out}}(\mathcal{A}^{\mathcal{S}_2} \mid \mathcal{S}_1(E)).$$

Basically, robust correctness implies, that in the honest-majority setting, the protocol will end in a way that each honest party P_i for $i \in I$ will hold a tuple (E_i, s_i) , for which there exists a polynomial $f(X) \in \mathbb{Z}_N[X]_t$, such that $E_i = [f(0)]E_0$ and $s_i = f(i)$, up to negligible probability. Note that the definition is for the case of a single PK, and it is straightforward to generalize it to the case of an extended (structured) PK. Secrecy on the other hand implies, that the adversary \mathcal{A} , controlling up to t malicious parties, cannot learn anything about s , other than what it can learn from the public key $E = [s]E_0$.

⁴ We emphasize however, that our definition of secrecy is the same as the original one introduced in [19] and thus differs from the “weaker” version presented in [6].

⁵ $D_{\text{out}}(\mathcal{A}^{\mathcal{O}} \mid \{P_i^{\mathcal{O}}\}_{i \in I}) = \{(A, E_{i^*}) \mid A, \{(E_i, s_i)\}_{i \in I} \leftarrow \langle \mathcal{A}^{\mathcal{O}} \mid \{P_i^{\mathcal{O}}\}_{i \in I} \rangle\}$ for any $i^* \in I$.

Piecewise Verifiable Proofs. Next, we revisit Piecewise Verifiable Proofs (PVPs), introduced in [6] to make the secret sharing step in CSI-RAShi verifiable. PVPs are ZK proofs for a list of relations R_0, \dots, R_n with the *same* witness space, where individual statements can be verified independently. For a list of statements x_0, \dots, x_n , a PVP allows one to prove the existence of a witness w such that $(x_i, w) \in R_i$ for any $i \in \{0, \dots, n\}$. The proof is of the form $(\tilde{\pi}, \{\pi_i\}_{i \in \{0, \dots, n\}})$, where $(\tilde{\pi}, \pi_0)$ allows verification of x_0 w.r.t. R_0 (called the *main* proof) and π_i for $i \in \{1, \dots, n\}$ further allows verification of x_i w.r.t. R_i . In particular, in [6], the witnesses constitute elements $f(X) \in \mathbb{Z}_N[X]_t$, i.e. polynomials in the variable X with coefficients defined over \mathbb{Z}_N and of degree at most t . The relations are given as

$$R_0 = \{((E_0, E_1), f(x)) \mid E_1 = [f(0)]E_0\} \wedge R_i = \{(x_i, f(x)) \mid f(i) = x_i\}, \quad (1)$$

for $i = 1, \dots, n$, and $E_0, E_1 \in \mathcal{E}$, i.e. supersingular elliptic curves defined over a prime field \mathbb{F}_p . Note that the witness is the same for all the relations.

CSI-RAShi [6] is an n -party honest-majority DKG protocol, that uses PVPs in its VSS step. The idea is that a party, called the dealer, is in possession of some polynomial $f(X) \in \mathbb{Z}_N[X]_t$, where $t \leq \lfloor \frac{n-1}{2} \rfloor$. The dealer publishes $E_1 = [f(0)]E_0$ and distributes the shares $f(i)$ to parties P_1, \dots, P_n . To prove that these shares are correct, the dealer publishes a PVP for the relations R_0, R_1, \dots, R_n as described above, and each party P_i can then verify the main statement R_0 as well as the statement R_i related to their share. In the honest-majority setting, if all honest parties agree that their share is correct, they know that they each possess a share of a unique polynomial $f(X)$ of degree t , whose evaluation in 0 is in the commitment $E_1 = [f(0)]E_0$.

We describe the proof generation and verification of their PVP scheme in Algorithms 1 and 2, where $\mathcal{H} : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ is a random oracle and $\mathcal{C} : \{0, 1\}^* \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$ is a commitment scheme that is collapsing [33, Definition 12] and quantum computationally hiding [6, Definition 2]. The authors prove that their PVP scheme is correct, sound and ZK, therefore constitutes a ZK proof system for the individual relations R_0, \dots, R_n . For more details, we refer the reader to the original CSI-RAShi paper [6].

The CSI-RAShi DKG Protocol. We quickly outline the building blocks and underlying security rationale of CSI-RAShi [6]. We refer the reader to the original source for more details and an algorithmic description of the protocol. The CSI-RAShi protocol consists of two consecutive phases, the VSS step, and the PK computation. The VSS is executed as outlined in Sect. 2.2, using a Pedersen-type approach [23] for a distributed secret generation. The shares are then verified using PVPs and any inconsistencies are resolved using the complaint resolution protocol introduced in [20]. At the end of the VSS, the parties that have not been disqualified, define the qualified set Q , which also implicitly defines the secret $s = \sum_{i \in Q} f^{(i)}(0)$. In the PK computation step, the parties within the qualified set then engage in a round-robin MPC protocol to compute the PK $E = [s]E_0$ as successive computations of the type $F_i = [f^{(i)}(0)]F_{i-1}$, starting at

Algorithm 1: PVP.P: The prover of non-interactive PVP [6].

Input : A witness polynomial $f(X) \in \mathbb{Z}_N[X]_{\leq t}$,
 a statement $x = ((E_0, E_1), x_1, \dots, x_n)$.

Output: A non-interactive piecewise proof π of the relations in equation (1).

for $j = 1, \dots, \lambda$ **do**
 $\lfloor b_j \leftarrow \mathbb{Z}_N[X]_{\leq t}$ uniformly at random; Set $\hat{E}_j \leftarrow [b_j(0)]E_0$

Sample $y_0, y'_0 \leftarrow \{0, 1\}^\lambda$ uniformly at random, and set
 $C_0 \leftarrow \mathcal{C}(\hat{E}_1 \parallel \dots \parallel \hat{E}_\lambda, y_0)$, and $C'_0 \leftarrow \mathcal{C}(E_0, E_1, y'_0)$.

for $i = 1, \dots, n$ **do**
 $\lfloor y_i, y'_i \leftarrow \{0, 1\}^\lambda$ uniformly at random and set $C_i \leftarrow \mathcal{C}(b_1(i) \parallel \dots \parallel b_\lambda(i), y_i)$
 \lfloor and $C'_i \leftarrow \mathcal{C}(x_i, y'_i)$

$\mathbf{d} = d_1 \dots d_\lambda \leftarrow \mathcal{H}(\mathbf{C}, \mathbf{C}')$, where $\mathbf{C} = (C_0, \dots, C_n)$, $\mathbf{C}' = (C'_0, \dots, C'_n)$

for $j = 1, \dots, \lambda$ **do**
 $\lfloor r_j(x) \leftarrow b_j(x) - d_j f(x) \pmod N$

return $\tilde{\pi} = (\mathbf{C}, \mathbf{C}', \mathbf{r})$ and $\{\pi_i = (y_i, y'_i)\}_{i=0}^n$, where $\mathbf{r} = (r_1, \dots, r_\lambda)$.

Algorithm 2: PVP.V: The verifier of non-interactive PVP [6].

Input : An index $i = 0, \dots, n$, a statement piece x_i of the form $x_0 = (E_0, E_1)$
 if $i = 0$, or $x_i \in \mathbb{Z}_N$ if $i \neq 0$, as well as a proof piece
 $(\tilde{\pi}, \pi_i) = ((\mathbf{C}, \mathbf{C}', \mathbf{r}), (y_i, y'_i))$.

Output: true or false

if $C'_i \neq \mathcal{C}(x_i, y'_i)$ **then**
 \lfloor **return** false

$d_1 \dots d_\lambda \leftarrow \mathcal{H}(\mathbf{C}, \mathbf{C}')$

if $i == 0$ **then**
 \lfloor **for** $j = 1, \dots, \lambda$ **do**
 \lfloor **if** $d_j == 0$ **then** $\tilde{E}_j \leftarrow [r_j(0)]E_0$
 \lfloor **else** $\tilde{E}_j \leftarrow [r_j(0)]E_1$
 \lfloor **return** $C_0 == \mathcal{C}(\tilde{E}_1, \dots, \tilde{E}_\lambda, y_0)$

else
 \lfloor **return** $C_i == \mathcal{C}(r_1(i) + d_1 x_i \parallel \dots \parallel r_\lambda(i) + d_\lambda x_i, y_i)$

$F_0 = E_0$ and ending at $F|_{Q_1} = E$. Parties prove correctness of their action using the ZK proofs introduced in [13, Sec. 3.1]. The share $f^{(i)}(0)$ of any party that misbehaves in the protocol can be reconstructed using Lagrange interpolation, so that the protocol always succeeds in the honest-majority setting.

CSI-RAShi satisfies the robust correctness property of Definition 2.2, relying on the soundness of PVPs. Furthermore, it satisfies a weaker variant of the secrecy property, which unlike Definition 2.3 assumes the input curve E to be uniformly sampled from \mathcal{E} , rather than being arbitrary [6, Definition 4]. This modification stems from the fact, that secrecy relies not only on the ZK property

of PVPs, but also on the isogeny-based Decisional Diffie-Hellman assumption (DDH) [32, Problem 2.2], which only holds for random inputs.⁶

Costs of CSI-RAShi. We discuss the computational and communication costs of the CSI-RAShi protocol. The authors of [6] state the total sequential cost of the protocol as $n + 2 + (4n + 1)\lambda$ isogeny computations, arguing that other costs are negligible in comparison. For direct comparison with the results in Tables 2 and 3, we summarize the costs of CSI-RAShi and its extended and structured versions in Table 1 below.

Table 1. Computational (top) and communication (bottom) cost of CSI-RAShi and its extended and structured versions. The computational costs in terms of isogenies are taken from [7] and [2], respectively. The communication cost for the basic protocol can be recovered from the extended case, by setting $k = 1$. Note that the factor $\frac{1}{2}$ in the C_C -communication cost comes from the λ -bit challenge in the zero-knowledge proofs of [7], which is half the size of a commitment scheme output ($C_C = 2\lambda$).

	Basic	Extended	Structured
T_E	$1 + 2n\lambda$	$(1 + 2n\lambda)k$	$1 + 2n\lambda$
T_I	$n + 2 + (4n + 1)\lambda$	$(n + 2)(k - 1) + 2n\lambda(3k - 1)$	$2 + n(k - 1) + 2\lambda(n(\sqrt{k - 1} + 1)^2 + 1)$
T_C	$6n - 2$	$(6n - 2)k$	$6n - 2$
T_H	$3n$	$3nk$	$n(2k + 1)$

	Extended	Structured
C_N	$k\lambda(t + 2)$	$\lambda(t + 1 + \sqrt{k - 1})$
C_E	$3k$	$k + 2$
C_C	$(3n + 2 + \frac{1}{2})k$	$3n + 2 + \frac{1}{2}\sqrt{k - 1}$

Security Assumptions. Next, we recall the security definitions of isogeny-based Decisional Diffie-Hellman assumption (DDH) [32, Problem 2.2], and $(c_0, c_1, \dots, c_{k-1})$ -Vectorization Problem with Auxiliary Inputs (\mathbb{C}_{k-1} -VPwAI) [3], that are used in CSI-RAShi [6] and its structured variant [2], respectively.

Definition 2.4 (Decisional Diffie-Hellman assumption). *Distinguish with non-negligible advantage between the distributions $([a]E, [b]E, [a + b]E)$ and $([a]E, [b]E, [c]E)$, where a, b and c are chosen uniformly at random from \mathbb{Z}_N .*

Definition 2.5 ($(c_0, c_1, \dots, c_{k-1})$ -Vectorization Problem with Auxiliary Inputs (\mathbb{C}_{k-1} -VPwAI)). *Given an element $E \in \mathcal{E}$ and the pairs $(c_i, [c_i x]E)_{i=1}^{k-1}$, where $\mathbb{C}_{k-1} = \{c_0 = 0, c_1 = 1, c_2, \dots, c_{k-1}\}$ is an exceptional set, find $x \in \mathbb{Z}_N$.*

⁶ We note that the authors of [6] use the description of *very hard homogeneous spaces* as introduced by Couveignes [12], where this assumption is called the decisional parallelization problem.

3 Robust DKG for CSIDH Using Bivariate Polynomials

In this section, we introduce a new DKG protocol for CSIDH, based on a VSS using bivariate polynomials [31], an idea that was originally introduced in [4]. When comparing to CSI-RAShI [6], the main advantage of this approach is that we can replace the piecewise verifiable proofs in the VSS step with simple comparison operations and further replace the ZK proofs used in the PK computation step with the cheaper piecewise verifiable proofs. This results in a more efficient DKG protocol. On the downside, the protocol has increased the communication cost, mainly due to the nature of the IT-secure VSS, and the number of corrupted parties is restricted to $t < \frac{n}{3}$, rather than honest-majority (i.e. $t < \frac{n}{2}$), as is the case for CSI-RAShI. Throughout this section, we assume that $n < N_1$, where N_1 is the smallest divisor of $N \mid \#\text{cl}(\mathcal{O})$.

3.1 A VSS Based on Bivariate Polynomials

We revisit the VSS approach from [1,31], using bivariate polynomials. This section follows closely along the lines of [1, Sec. 5]. Yet, since we are working with polynomials over rings, we have to slightly adapt the theorems and point out differences in the proofs, in order to account for this fact.

A bivariate polynomial $S(X, Y) \in \mathbb{Z}_N[X, Y]$ of degree t is a polynomial over variables X and Y , each of which has degree at most t , i.e. we can write it as

$$S(X, Y) = \sum_{i=0}^t \sum_{j=0}^t a_{ij} X^i Y^j.$$

We write $S(X, Y) \in \mathbb{Z}[X, Y]_t$. The idea behind using bivariate polynomials in the VSS of [4] is related to the following Theorem, adapted from [1, Claim 5.2].

Theorem 3.1. *Let $f_1(X), \dots, f_{t+1}(X) \in \mathbb{Z}_N[X]$ be polynomials of degree t . Then there exists a unique bivariate polynomial $S(X, Y) \in \mathbb{Z}_N[X, Y]$ of degree t such that for every $k = 1, \dots, t + 1$, it holds that $S(X, k) = f_k(X)$.*

Proof. The proof is identical to the proof of Claim 5.2 of [1], which uses Lagrange interpolation over a finite field to construct $S(X, Y)$ from the evaluations $S(X, k)$. Since $t + 1 < N_1$, the Lagrange interpolation polynomials are well-defined over the ring \mathbb{Z}_N and the proof works analogously. \square

Thus every degree- t bivariate polynomial can be interpolated from $t+1$ univariate degree- t polynomials in the same way that every degree- t univariate polynomial can be interpolated from $t + 1$ points. Conversely, the evaluation of a bivariate polynomial in one of its variables defines a univariate polynomial (of the same degree), in the same way as evaluating a univariate polynomial defines a point. Similarly, if less than $t + 1$ univariate polynomials are known, the overarching bivariate polynomial is IT hidden as the following theorem suggests.

Theorem 3.2. *Let $I \subset \{1, \dots, n\}$ with $|I| \leq t$ and let $f, g \in \mathbb{Z}_N[X]_t$ with $f(i) = g(i)$ for all $i \in I$. Then for any two bivariate polynomials $S_1(X, Y), S_2(X, Y) \in \mathbb{Z}_N[X, Y]_t$ with $S_1(0, Y) = f(Y)$ and $S_2(0, Y) = g(Y)$, the two sets*

$$\{i, S_1(X, i), S_1(i, Y)\}_{i \in I} \quad \text{and} \quad \{i, S_2(X, i), S_2(i, Y)\}_{i \in I}$$

are indistinguishable.

Proof. See proof of Claim 5.4 of [1]. □

Bivariate polynomials can be used in VSSs as follows [4]. To share a secret x between n parties P_1, \dots, P_n , one party, called the *dealer*, samples a random polynomial $q(Z) \in \mathbb{Z}_N[Z]_t$ such that $q(0) = x$. Then the dealer samples a random bivariate polynomial $S(X, Y) \in \mathbb{Z}_N[X, Y]_t$ with the constraint that $S(0, Z) = q(Z)$, so in particular $S(0, 0) = x$. In order to distribute x among the n parties, the dealer sets

$$f_i(X) = S(X, i) \quad \text{and} \quad g_i(Y) = S(i, Y),$$

then sends $f_i(X)$ and $g_i(Y)$ privately to party P_i for $i = 1, \dots, n$. Each party can now construct their share as $x_i = f_i(0) = S(0, i)$, which allows to recover $x = S(0, 0)$ via Lagrange interpolation. The true advantage of using bivariate polynomials manifests itself in the share verification step, which relies on the following theorem.

Theorem 3.3. *Let $K \subseteq \{1, \dots, n\}$ be a set of indices with $|K| \geq t + 1$ and let $\{f_k(X), g_k(X)\}_{k \in K}$ be a set of pairs of polynomials in $\mathbb{Z}_N[X]_t$. If $f_i(j) = g_j(i)$ holds for every $i, j \in K$, then there exists a unique bivariate polynomial $S(X, Y) \in \mathbb{Z}_N[X, Y]_t$, such that for every $k \in K$, $f_k(X) = S(X, k)$ and $g_k(Y) = S(k, Y)$.*

Proof. This proof works analogous to the proof of Claim 5.3 of [1], by using the uniqueness of $S(X, Y)$ guaranteed by Theorem 3.1 (instead of [1, Claim 5.2]). □

Thus, in order to verify the correctness of their shares, each pair of parties P_i, P_j simply checks that $f_i(j) = g_j(i)$ and $g_i(j) = f_j(i)$. If all these tests succeed, then parties know that their shares are consistent with a single bivariate polynomial and that the sharing was successful. In the converse case, if some of these checks do not succeed, the players engage in the following steps to resolve the conflict [1].

1. If for a player P_i , the checks $f_i(j) \stackrel{?}{=} g_j(i)$ or $g_i(j) \stackrel{?}{=} f_j(i)$ do not succeed, then P_i broadcasts a complaint by disclosing $(i, j, f_i(j), g_i(j))$. As a response, the dealer reveals $(i, f_i(X), g_i(Y))$. Then each other player P_k evaluates the complaints as below. Whenever players are satisfied with the complaint resolution, they broadcast consistent, otherwise they don't.
 - (a) If, for every *joint complaint*, e.g. $(i, j, f_i(j), g_i(j))$ by P_i and $(j, i, f_j(i), g_j(i))$ by P_j , the dealer does not reveal $(i, f_i(X), g_i(Y))$ nor $(j, f_j(X), g_j(Y))$, jump to step 2. (without broadcasting consistent), otherwise continue.

- (b) If there exists a response $(k, f_k(X), g_k(X))$, P_k accepts this as its new shares, then jumps to step 2. (without broadcasting consistent), otherwise continue.
 - (c) For any other response, verify if the polynomials revealed by the dealer indeed do not satisfy the necessary checks. If they don't, jump to step 2. (without broadcasting consistent), otherwise continue.
 - (d) Broadcast the message consistent.
2. If at least $n - t$ parties have broadcasted consistent, then each party outputs its share $x_i = f_i(0)$, otherwise return \perp .

Theorem 3.4. *For $t < n/3$, the secret sharing protocol, along with the conflict resolution procedure described above, implements F_{VSS} in a correct and secure way, for a static malicious adversary (which might include the dealer).*

Proof. We refer the reader to the proof of Theorem 5.7 of [1]. The proof here works the same way, except that references to Claims 5.3 and 5.4 should be substituted with Theorems 3.3 and 3.2, respectively. Furthermore, the values $\alpha_1, \dots, \alpha_n$, at which the polynomials are evaluated in these proofs, should be chosen from the subset $\{1, \dots, N_1 - 1\}$. For simplicity, we choose $\{1, \dots, n\}$. \square

3.2 Robust Distributed Generation of $[x]E_0$

We can easily extend the protocol described in the previous section to a distributed VSS by using the approach from [23]. There, each party P_i involved in the protocol individually takes the role of the dealer and constructs and distributes shares of its secret $x^{(i)}$ to each other party. The final secret is then simply the sum of all the secrets and each party's share is the sum of all the shares. Yet, if parties misbehave (as dealers or as receiving parties), the checks described in the previous section will uncover this and parties will be disqualified. In the end, the secret and secret shares are then actually defined as the sum of the respective elements of all the *qualified players*. After a shared secret x was implicitly defined in the *VSS step*, the parties engage in a *PK computation step* to compute the PK $[x]E_0$. Thus our DKG protocol's steps follow a two step approach, along the lines of [20] or [6]. In contrast to those protocols however, the parties do not have to commit to their shares in the first phase, but rather resort to the pairwise comparisons described in the previous subsection. We present our protocol in Fig. 1 and prove the following theorem in the full version of the paper.

Theorem 3.5. *If PVPs are sound, then the DKG protocol of Fig. 1 is correct and robust. If PVPs are zero-knowledge, then the DKG protocol satisfies the secrecy property.*

Below, we explain some details of the steps in Fig. 1.

VSS Step. We assume each party P_i wants to share a secret $x^{(i)}$. The distribution of $x^{(i)}$ is achieved by sampling a random univariate polynomial $q^{(i)}(Z) \in \mathbb{Z}_N[Z]_t$ with $x^{(i)} = q^{(i)}(0)$ and a random bivariate polynomial $S^{(i)}(X, Y) \in \mathbb{Z}_N[X, Y]_t$ with $S^{(i)}(0, Z) = q^{(i)}(Z)$. Then P_i sets

$$f_j^{(i)}(X) = S^{(i)}(X, j) \quad \text{and} \quad g_j^{(i)}(Y) = S^{(i)}(j, Y)$$

for $j = 1, \dots, n$ and sends $f_j^{(i)}(X)$ and $g_j^{(i)}(Y)$ privately to party P_j for all j . The parties engage in the protocol to verify the correctness of their shares. In case the checks fail, i.e. \perp is returned while a player P_k is the dealer, then P_k will be disqualified and the protocol continues without the inputs by P_k . Thus, after

Verifiable Secret Sharing:

1. For $i = 1, \dots, n$, player P_i
 - (a) samples $q^{(i)}(Z) \leftarrow \mathbb{Z}_N[Z]_t$ and sets $x^{(i)} = q^{(i)}(0)$,
 - (b) samples $S^{(i)}(X, Y) \leftarrow \mathbb{Z}_N[X, Y]_t$ with $S^{(i)}(0, Z) = q^{(i)}(Z)$,
 - (c) for $j = 1, \dots, n$, defines $f_j^{(i)}(X) = S^{(i)}(X, j)$ and $g_j^{(i)}(Y) = S^{(i)}(j, Y)$ and sends $\{f_j^{(i)}(X), g_j^{(i)}(Y)\}$ privately to party P_j .
2. For $k = 1, \dots, n$, each pair of players P_i, P_j checks that $f_i^{(k)}(j) = g_j^{(k)}(i)$ and $g_i^{(k)}(j) = f_j^{(k)}(i)$. Whenever one of these checks fails, the concerned player runs the conflict resolution procedure described in Sec. 3.1. In case the procedure outputs \perp , the dealer P_k of the concerned polynomials is disqualified, otherwise the protocol continues normally.
3. In the end, all the honest players agree on the same set of qualified players $Q \subseteq \{1, \dots, n\}$, and the shared secret key x is given as the sum of the individual secrets of the qualified players $x = \sum_{i \in Q} x^{(i)}$, while the parties' shares of x can be constructed as $x_j = \sum_{i \in Q} f_j^{(i)}(0)$.

Computing the Public Key:

4. Let for simplicity $Q = \{1, \dots, n'\}$. In a round-robin way, the qualified players now compute $F_i \leftarrow [x^{(i)}]F_{i-1}$ where $F_0 = E_0$. At each step, using Algorithm 1, player P_i further creates and publishes a PVP proof

$$\pi^{(i)} = (\tilde{\pi}^{(i)}, \pi_1^{(i)}, \dots, \pi_{n'}^{(i)}) \leftarrow \text{PVP.P}((F_{i-1}, F_i); q^{(i)}(Z))$$

which includes a main proof $(\tilde{\pi}^{(i)}, \pi_0^{(i)})$ as well as individual proof pieces $\pi_j^{(i)}$ for each other player P_j .

5. Using Algorithm 2, each other player P_j verifies both $\text{PVP.V}(j, f_j^{(i)}(0), \tilde{\pi}^{(i)}, \pi_j^{(i)})$ and $\text{PVP.V}(0, (F_{i-1}, F_i), \tilde{\pi}^{(i)}, \pi_0^{(i)})$. Whenever a verification of $\pi^{(i)}$ fails, the verifier P_j broadcasts $f_j^{(i)}(X)$. All other parties verify correctness of $f_j^{(i)}(X)$ as in step 2. If it is correct, since there are at least $t + 1$ honest players, they will be able to reconstruct $q^{(i)}(0)$, compute F_i and proceed with the protocol (and potentially disqualify P_i). Otherwise, if the checks of $f_j^{(i)}(0)$ fail, the complaint can be ignored (or P_j disqualified). In the latter case, the shares of P_j can also be reconstructed by the at least $t + 1$ honest players.
6. At the end of the round-robin, the parties return the public key $F_{n'} = [x]E_0$.

Fig. 1. The DKG protocol for a single public key $[x]E_0$.

the consistency check step, there will be a set $Q \subseteq \{1, \dots, n\}$ of qualified parties which implicitly defines the shared secret x as

$$x = \sum_{i \in Q} x^{(i)} = \sum_{i \in Q} S^{(i)}(0, 0) = \sum_{i \in Q} q^{(i)}(0) = q(0).$$

Each party can derive their share x_j of x as

$$x_j = q(j) = \sum_{i \in Q} q^{(i)}(j) = \sum_{i \in Q} f_j^{(i)}(0),$$

and $x = q(0)$ can be recovered by any subset of at least $t + 1$ parties.

PK Computation Step. In this step, the parties in Q engage in a round-robin protocol for computing the public key $[x]E_0$. For simplicity, we assume $Q = \{1, \dots, n'\}$. Then, at step i , party P_i will compute

$$F_{i-1} \mapsto F_i = [x^{(i)}]F_{i-1},$$

where F_0 is some starting curve. At the end of the round-robin,

$$F_{n'} = [x^{(n')}] \dots [x^{(1)}]F_0 = \left[\sum_{i \in Q} x^{(i)} \right] F_0 = [x]F_0$$

is the public key. The only thing left to do for each player P_i is to prove that they used the correct $x^{(i)} = q^{(i)}(0)$. Since the other parties P_j each possess a share $f_j^{(i)}(0) = S^{(i)}(0, j) = q^{(i)}(j)$, the dealer P_i can convince them of having used the correct action by proving the following relation

- $F_i = [q^{(i)}(0)]F_{i-1}$ and
- for $j \in Q$, player P_j possesses the share $q^{(i)}(j)$ of $q^{(i)}(0)$.

for the witness polynomial $q^{(i)}(Z) \in \mathbb{Z}_N$. It turns out that this is exactly the language that is proved using the PVPs from Sect. 2.2, originally introduced in [6]. Thus a player P_i can convince the other players of having acted with $q^{(i)}(0)$, by relating it to the shares $q^{(i)}(j) = f_j^{(i)}(0)$ distributed in the VSS step.

Extended Public Keys. The DKG protocol in Fig. 1 can be easily extended to sample k public keys, which is required in protocols like CSI-FiSh [7]. This can simply be done by generating multiple polynomials $S_1^{(i)}, \dots, S_k^{(i)}$ for the respective secrets $x^{(1,i)}, \dots, x^{(k,i)}$. In the PK computation step, parties then compute the curves $F_i^1 \leftarrow [x^{(1,i)}]F_{i-1}^1, \dots, F_i^k \leftarrow [x^{(k,i)}]F_{i-1}^k$, and prove correctness at each step. In this case, it requires running k independent PVPs.

4 Robust DKG for Structured Public Keys

We recall that for a given secret $x \in \mathbb{Z}_N$, an SPK [2,3] has the form $\{E_i = [c_i x]E_0\}_{i=1}^k$, where $c_i \in \mathbb{C}_k$ and $\mathbb{C}_k = \{c_1 = 1, c_2, \dots, c_k\}$ is a public (super)exceptional set, see also Sect. 2.1. In this section, we present a new variant of the DKG protocol in Fig. 1, which would allow a set of parties to sample an SPK in a distributed manner.

VSS Step. Since an SPK has only one secret key, we need to execute the VSS step only once. This is done exactly as in Fig. 1.

SPK Computation Step. In the SPK computation step, for a given superexceptional set $\mathbb{C}_k = \{c_1 = 1, c_2, \dots, c_k\}$, each party P_i has to compute

$$F_i^1 \leftarrow [x^{(i)}]F_{i-1}^1, F_i^2 \leftarrow [c_2x^{(i)}]F_{i-1}^2, \dots, F_i^k \leftarrow [c_kx^{(i)}]F_{i-1}^k,$$

and give a proof that they updated all the curves in the SPK with correct factors of the secret key $x^{(i)} = q^{(i)}(0)$ shared with other parties. Since the other parties P_j each possess a share $f_j^{(i)}(0) = S^{(i)}(0, j) = q^{(i)}(j)$ from the VSS step, party P_i needs to convince them that it used the correct action by proving the following relation:

- For a public superexceptional set $\mathbb{C}_k = \{c_1 = 1, c_2, \dots, c_k\}$:
 $F_i^1 = [c_1q^{(i)}(0)]F_{i-1}^1 \wedge \dots \wedge F_i^k = [c_kq^{(i)}(0)]F_{i-1}^k$, and
- for $j \in \mathcal{Q}$, player P_j possesses the share $q^{(i)}(j)$ of $q^{(i)}(0)$,

where $q^{(i)}(Z) \in \mathbb{Z}_N$ is the witness polynomial. To prove the above relations, we propose a new PVP scheme, which we call Structured PVP (SPVP), that can be considered as an extension of the one proposed in [6, Algorithms 3 and 4], and which will allow us to prove the computation of SPKs in our DKG.

Structured PVP (SPVP). We define the following list of relations $R = (R_0, \dots, R_n)$, whose common witness space is $\mathbb{Z}_N[X]_{\leq t}$, the set of polynomials over \mathbb{Z}_N of degree at most t :

$$R_0 = \{((\mathbb{C}_k, F_1, F'_1, \dots, F_k, F'_k), f(x)) \mid (F'_l = [c_l f(0)]F_l)_{l=1}^k\},$$

$$\forall i = 1, \dots, n : R_i = \{(x_i, f(x)) \mid f(i) = x_i\}. \tag{2}$$

A statement x_0 for R_0 consists of a public superexceptional set $\mathbb{C}_k = \{c_1 = 1, c_2, \dots, c_k\}$, and a set of curves $(F_1, F'_1, \dots, F_k, F'_k) \in \mathcal{E}^{2k}$. Just as in the original PVPs, a statement x_i for the relations $\{R_i\}_{i=1, \dots, n}$ is an element of \mathbb{Z}_N .

The description of non-interactive SPVP for relations of the above form are presented in Algorithms 3 and 4. The algorithms use a random oracle $\mathcal{H} : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$, and a non-interactive commitment scheme $\mathcal{C} : \{0, 1\}^* \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$, where λ is the security parameter. In Appendix A, we prove the following Theorem.

Theorem 4.1. *Assuming that the commitment scheme \mathcal{C} is collapsing and quantum computationally hiding, the described non-interactive PVP for structured public keys (in Algorithms 3 and 4) is complete, sound, and ZK in the QROM for the list of relations given in Eq. (2).*

Efficiency. As it can be seen, by running $\text{SPVP}.P$ a party needs to compute λk group actions, query once to the random oracle \mathcal{H} , and $2(n + 1)$ times to the commitment scheme \mathcal{C} . Similarly, by running $\text{SPVP}.V$ a verifier needs to query once to the random oracle \mathcal{H} , and twice to the commitment scheme \mathcal{C} , and only in the case $i = 0$ it needs to compute λk group actions.

Ternary Challenge Space. The challenge space of the (structured) PVP schemes is binary. As a result, the PVP scheme constructed in [6] and its new variant proposed above need to be repeated λ times to achieve the soundness error $\frac{1}{2^\lambda}$. Unlike in CSI-RAShI, in our DKG protocols, the first party generates a PVP proof starting from the base curve E_0 . Since the class group enjoys a symmetry around the elliptic curve E_0 , this allows us to use the twist trick introduced in [7] and extend the challenge space of the PVP schemes to $\{-1, 0, 1\}$, with minimal changes to the descriptions in Algorithms 3 and 4. The resulting soundness

Algorithm 3: SPVP.P: The prover of non-interactive Structured PVP.

Input : A witness polynomial $f(X) \in \mathbb{Z}_N[X]_{\leq t}$,
a statement $x = ((\mathbb{C}_k, F_1, F'_1, \dots, F_k, F'_k), x_1, \dots, x_n)$.

Output: A non-interactive piecewise proof π of the relations in equation (2).

Parse $\mathbb{C}_k = \{c_1, c_2, \dots, c_k\}$.

for $j = 1, \dots, \lambda$ **do**

- └ $b_j \leftarrow \mathbb{Z}_N[X]_{\leq t}$ uniformly at random
- └ $\hat{F}_j^1 \leftarrow [c_1 b_j(0)]F_1, \dots, \hat{F}_j^k \leftarrow [c_k b_j(0)]F_k$

Sample $y_0, y'_0 \leftarrow \{0, 1\}^\lambda$ uniformly at random, and set

$\mathbb{C}_0 \leftarrow \mathcal{C}(\hat{F}_1^1, \dots, \hat{F}_1^k \parallel \dots \parallel \hat{F}_\lambda^1, \dots, \hat{F}_\lambda^k, y_0)$,

$\mathbb{C}'_0 \leftarrow \mathcal{C}(F_1, F'_1 \parallel \dots \parallel F_k, F'_k, y'_0)$.

for $i = 1, \dots, n$ **do**

- └ $y_i, y'_i \leftarrow \{0, 1\}^\lambda$ uniformly at random; set $\mathbb{C}_i \leftarrow \mathcal{C}(b_\lambda(i) \parallel \dots \parallel b_\lambda(i), y_i)$; and
- └ $\mathbb{C}'_i \leftarrow \mathcal{C}(x_i, y'_i)$

$\mathbf{d} = d_1 \dots d_\lambda \leftarrow \mathcal{H}(\mathbb{C}, \mathbb{C}')$, where $\mathbb{C} = (\mathbb{C}_0, \dots, \mathbb{C}_n)$, $\mathbb{C}' = (\mathbb{C}'_0, \dots, \mathbb{C}'_n)$

for $j = 1, \dots, \lambda$ **do**

- └ $r_j(x) \leftarrow b_j(x) - d_j f(x) \pmod N$

return $\tilde{\pi} = (\mathbb{C}, \mathbb{C}', \mathbf{r})$ and $\{\pi_i = (y_i, y'_i)\}_{i=0}^n$, where $\mathbf{r} = (r_1, \dots, r_\lambda)$.

Algorithm 4: SPVP.V: The verifier of non-interactive Structured PVP.

Input : An index $i = 0, \dots, n$, a statement piece x_i of the form
 $x_0 = (\mathbb{C}_k, F_1, F'_1, \dots, F_k, F'_k)$ if $i = 0$, or $x_i \in \mathbb{Z}_N$ if $i \neq 0$, as well as a
proof piece $(\tilde{\pi}, \pi_i) = ((\mathbb{C}, \mathbb{C}', \mathbf{r}), (y_i, y'_i))$.

Output: true or false

if $\mathbb{C}'_i \neq \mathcal{C}(x_i, y'_i)$ **then**

- └ **return** false

$d_1 \dots d_\lambda \leftarrow \mathcal{H}(\mathbb{C}, \mathbb{C}')$

if $i == 0$ **then**

- └ **for** $j = 1, \dots, \lambda$ **do**
- └ └ **if** $d_j == 0$ **then** $\tilde{F}_j^1 \leftarrow [c_1 r_j(0)]F_1, \dots, \tilde{F}_j^k \leftarrow [c_k r_j(0)]F_k$
- └ └ **else** $\tilde{F}_j^1 \leftarrow [c_1 r_j(0)]F'_1, \dots, \tilde{F}_j^k \leftarrow [c_k r_j(0)]F'_k$
- └ **return** $\mathbb{C}_0 == \mathcal{C}(\tilde{F}_1^1, \dots, \tilde{F}_1^k \parallel \dots \parallel \tilde{F}_\lambda^1, \dots, \tilde{F}_\lambda^k, y_0)$

else

- └ **return** $\mathbb{C}_i == \mathcal{C}(r_1(i) + d_1 x_i \parallel \dots \parallel r_\lambda(i) + d_\lambda x_i, y_i)$

error rate becomes $\frac{1}{3}$ and the number of repetitions reduces to $\lambda' := \lceil \lambda / \log_2 3 \rceil$ to achieve the soundness error $\frac{1}{2^\lambda}$. This results in a noticeable gain, especially if the number of parties is small.

Construction of the DKG Protocol. Figure 2 describes the construction of our proposed robust DKG protocol for structured public keys. The protocol uses the distributed VSS scheme described in Sect. 3.1, and the non-interactive Structured PVP (SPVP.P, SPVP.V), given in Algorithms 3 and 4, as a subroutine. The proof of the following theorem can be found in the full version of the paper.

Theorem 4.2. *If structured PVPs are sound, then the DKG protocol of Fig. 2 to generate structured public keys is correct and robust. If structured PVPs are ZK, then the DKG protocol satisfies the secrecy property.*

Verifiable Secret Sharing:
 This is done in the same way as Fig. 1. The players furthermore agree on an exceptional set $\mathbb{C}_k = \{c_1 = 1, c_2, \dots, c_k\}$.

Computing the Structured Public Key:

4. Let for simplicity $Q = \{1, \dots, n'\}$. Given a superexceptional set $\mathbb{C}_k = \{c_1 = 1, c_2, \dots, c_k\}$, a qualified set of parties engage in a round-robin protocol, and party P_i computes

$$F_i^1 \leftarrow [x^{(i)}]F_{i-1}^1, F_i^2 \leftarrow [c_2x^{(i)}]F_{i-1}^2, \dots, F_i^k \leftarrow [c_kx^{(i)}]F_{i-1}^k,$$
 where $F_0^1 = F_0^2 = \dots = F_0^k = E_0$. At each step, player P_i further uses the non-interactive Structured PVP (SPVP.P, SPVP.V), given in Algorithms 3 and 4, and creates and publishes a structured PVP proof,

$$\pi^{(i)} = (\tilde{\pi}^{(i)}, \pi_1^{(i)}, \dots, \pi_{n'}^{(i)}) \leftarrow \text{SPVP.P}(\mathbb{C}_k, (F_{i-1}^1, F_i^1, \dots, F_{i-1}^k, F_i^k); q^{(i)}(Z))$$
 which includes a main proof $(\tilde{\pi}^{(i)}, \pi_0^{(i)})$ as well as individual proof pieces $\pi_j^{(i)}$ for each other player P_j .
5. Each other player P_j verifies both $\text{SPVP.V}(j, f_j^{(i)}(0), \tilde{\pi}^{(i)}, \pi_j^{(i)})$ and $\text{SPVP.V}(0, (\mathbb{C}_k, (F_{i-1}^1, F_i^1, \dots, F_{i-1}^k, F_i^k)), \tilde{\pi}^{(i)}, \pi_0^{(i)})$. Whenever a verification of $\pi^{(i)}$ fails, the verifier P_j broadcasts $f_j^{(i)}(X)$. All other parties verify correctness of $f_j^{(i)}(X)$ as in step 2 of Fig. 1. If it is correct, since there are at least $t + 1$ honest players, they will be able to reconstruct $q^{(i)}(0)$, compute F_i and proceed with the protocol (and potentially disqualify P_i). Otherwise, if the checks of $f_j^{(i)}(0)$ fail, the complaint can be ignored (or P_j disqualified). In the latter case, the shares of P_j can also be reconstructed by the at least $t + 1$ honest players.
6. At the end of the round-robin, the parties return the *structured* public key

$$F_{n'}^1 = [x]E_0, F_{n'}^2 = [c_2x]E_0, \dots, F_{n'}^k = [c_kx]E_0.$$

Fig. 2. The DKG protocol for structured public keys.

5 Efficiency of the DKG Protocols and Optimizations

We summarize the computational and communication costs of our DKG protocols in Tables 2 and 3. We express the computational cost as the *sequential* runtime of the protocol steps, i.e. the total runtime from start to finish, including when some of the parties are idle. The communication cost is expressed in terms of *outgoing* communication per party. These costs are established in detail in the Appendix B, where we also discuss optimizations in order to minimize them.

Comparison of Extended and Structured Cases. We end this section with a comparison between DKGs building extended or structured public keys in terms of computational and communication costs. A direct comparison of Tables 2 and 3 reveals that while extended DKGs scale linearly with k in every term, structured DKGs only scale with k in the number of isogeny computations T_I and the number of shared elliptic curves C_E . We summarize the trends for varying k and n in Figs. 3 and 4 below, and discuss more details in the Appendix B. Figs. 3 and 4 also compare our results with CSI-RAShi [6] with extended public keys (using the optimizations from [2, Sec. 4.3]), as well as the recently proposed *structured CSI-RAShi*

Table 2. Computational costs of the basic, extended and structured DKG in terms of polynomial evaluations T_E , isogeny computations T_I and calls to the commitment scheme T_C and random oracle T_H . The cost represents the total sequential cost of the protocol, including idle times by the parties. For compactness, we do not consider the twist trick described in the previous section; it can easily be reinstated by substituting the terms of the form $n\lambda$ to $\lambda' + (n - 1)\lambda$ in the factors of T_E and T_I . The impact of the twist trick is further discussed in Appendix B. We define $\chi_{n,k} = \lceil \frac{k}{n} \rceil - \lfloor \frac{k}{n} \rfloor$.

	Basic DKG	Extended DKG	Structured DKG
T_E	$2(n - 1)^2 + n\lambda(n + 2)$	$2(n - 1)^2k + n\lambda(n \lceil \frac{k}{n} \rceil + k + \chi_{n,k})$	$2(n - 1)^2 + n\lambda(2n + 1)$
T_I	$2n\lambda + n$	$n\lambda(k + \chi_{n,k}) + n \lceil \frac{k}{n} \rceil$	$n\lambda(k + \chi_{n,k}) + n \lceil \frac{k}{n} \rceil$
T_C	$2n(n + 3)$	$2n(\lceil \frac{k}{n} \rceil (n - 1) + 2k + 2\chi_{n,k})$	$2n(3n - 1)$
T_H	$2n$	$n(k + \chi_{n,k})$	n^2

Table 3. Communication costs of the extended and structured DKG in terms of the information contained in elements of \mathbb{Z}_N and \mathcal{E} , i.e. C_N and C_E respectively, and of the output of our commitment scheme C_C . The cost represents the outgoing cost per party. The cost of the basic DKG immediately follows by setting $k = 1$ in either case. We again ignore the twist trick for compactness, which we regain by the substitution $n\lambda \mapsto \lambda' + (n - 1)\lambda$ in the costs of C_N .

	Extended DKG	Structured DKG
C_N	$2k(n - 1)(n - t + 1) + kn\lambda(t + 1)$	$2(n - 1)(n - t + 1) + n\lambda(t + 1)$
C_E	nk	nk
C_C	$nk(3n + 2)$	$n(3n + 2)$

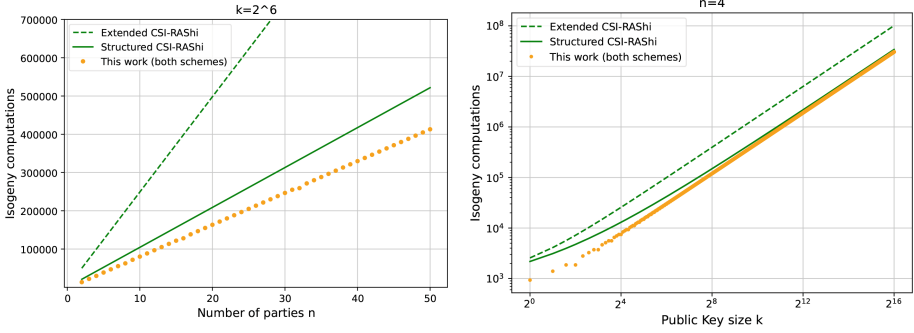


Fig. 3. Computational cost in terms of isogeny computations for the CSIDH-512 parameter set, shown as a function of the number of parties n (left; for $k = 2^6$) and as a function of the public key size k (right; for $n = 4$). We can see that our protocols generally outperform the protocols from [2, 6], most notably for large n or low k . For asymptotically large k , our protocols coincide with structured CSI-RASHi in the number of isogeny computations.

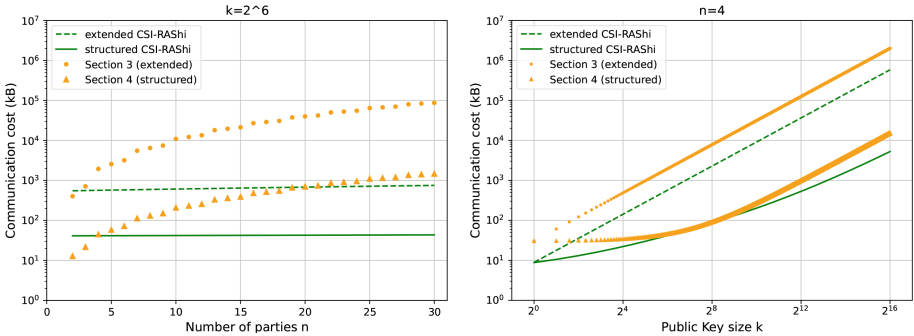


Fig. 4. Communication cost in kilobytes for the CSIDH-512 parameter set, shown as a function of the number of parties n (left; exemplified for $k = 2^6$) and as a function of the public key size k (right; for $n = 4$). We choose t as the largest integer $< n/3$. We compare our results to the extended and structured CSI-RASHi from [2]. The communication cost of our schemes is generally higher, but come close for small n and specific ranges of k . Notably, our structured PK scheme outperforms structured CSI-RASHi for $n = 2$ and $k < 2^{13}$ and for $n = 3$ in the parameter range $k \in [2^2, 2^{10}]$.

from [2, Sec. 5.3].⁷ Both of these schemes are honest-majority Shamir secret sharing based DKG protocols in the CSIDH setting. To our current knowledge, Structured CSI-RASHi represents the most efficient isogeny-based DKG in the literature for (structured) public keys of size $k > 1$.

⁷ We note that [2] also analyzes extended and structured versions of the Sashimi DKG [13], which is a full-threshold DKG. The authors in [2] show that the communication and computational costs of this DKG are basically the same as for CSI-RASHi, up to some barely noticeable constant factors. We therefore omit their analysis here.

6 Conclusion

In this paper, we presented two efficient robust DKG protocols in the CSIDH setting, based on secret sharing with bivariate polynomials, which outperform current alternatives [2, 6] in terms of computational cost.

The first protocol allows a set of parties to sample a public key like $[x]E_0$ or $([x_1]E_0, \dots, [x_k]E_0)$ and obtain a Shamir share of each of the secret keys. Such public keys are used in isogeny-based cryptosystems (e.g. [17, 26]) and signature schemes (e.g. [7]), respectively. The second protocol allows a set of parties to sample a structured public key, e.g. a PK of the form $([c_1x]E_0, \dots, [c_kx]E_0)$, as used in the CSI-SharK signature scheme [2], and obtain a Shamir share of x . Both protocols are secure in the QROM and achieve IT security in the VSS step. However, compared to current alternatives, our protocols generally require more communication between parties (except for some parameters) and a higher number of honest participants (more than two-thirds) during the VSS step.

In Sect. 5 we showed that for generating a single PK, $[x]E_0$, our first protocol is at least two times faster than the state-of-the-art scheme CSI-RAShI [6]. In cases where the number of parties is small, the improvement is even higher, e.g. for $n = 2$, our first DKG protocol is about 3 times faster and also requires 27% less communication. When generating extended or structured public keys, our protocols are also faster than the currently most efficient ones [2, 6]. For instance, using our protocol, 4 parties can sample a PK of size 4 kB (i.e., $k = 2^6$), for CSI-FiSh [7] and CSI-SharK [2], respectively 3.4 and 1.7 times faster than the current DKG protocols.

As a building block for our second DKG protocol, in Sect. 4, we presented a new PVP scheme specifically for SPKs, which we think can be of independent interest for future protocols using SPK.

Acknowledgments. This work has been supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract No. HR001120C0085, by the FWO under an Odyssey project GOH9718N, by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (Grant agreement No. 101020788 - Adv-ERC-ISOCRYPT), by CyberSecurity Research Flanders with reference number VR20192203, by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), by the Spanish Government under project PRODIGY (TED2021-132464B-I00), and by the Madrid Regional Government under project BLOQUES (S2018/TCS-4339). The last two projects are co-funded by European Union EIE, and Next Generation EU/PRTR funds.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ERC, DARPA, the US Government, the Spanish Government, Cyber Security Research Flanders or the FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

A Security Proofs

A.1 Proof of Theorem 4.1

We prove the security of new non-interactive Structured PVP, described in Algorithms 3 and 4 by the following theorem.

Theorem 4.1. *Assuming that the commitment scheme \mathcal{C} is collapsing and quantum computationally hiding, the described non-interactive PVP for the structured public keys (in Algorithms 3 and 4) is complete, sound, and ZK in the QROM for the list of relations given in Eq. (2).*

Proof. The proof of this theorem is analogous to the proof of [6, Theorem 2], with a few differences, which we will highlight in this proof.

A key peculiarity of PVPs (and SPVPs) is that they use a *weak* version of the Fiat–Shamir transform, i.e. where the random oracle is called with commitments as inputs rather than commitments and statements. In [6], the consequences regarding the security of this are treated in detail and PVPs could be proven secure, even with this modification. We refer the reader to [6, App. A] for more details. We note that these results also still apply to our case, so we will omit proving them again. Rather, let us point out, where the differences between SPVPs and PVPs lie. These are mainly in the definition of the relation R_0 , which in the original CSI-RAShI paper is defined as

$$\{(x_0 = (F_1, F'_1), f(x)) \mid (F'_1 = [f(0)]F_1)\},$$

and thus represents the special case $k = 1$ and $c_1 = 1$ of the definition in Eq. (2).⁸ Similarly, the commitments as represented in Algorithms 3 and 4 also reduce to the case $k = 1$.

As a result of this different structure, the proofs for completeness and soundness are adapted below for the case $i = 0$. The case $i \neq 0$ remains unchanged, as also here the relation is unchanged. We note that zero-knowledge immediately follows from the properties of the commitment scheme \mathcal{C} and is therefore analogous to the proof in [6].

Completeness. For any $j = 1, \dots, \lambda$, if $d_j = 0$, then $r_j = b_j$ and hence $\tilde{F}_j^l = [c_l r_j(0)]F_l = [c_l b_j(0)]F_l = \hat{F}_j^l$ for $l = 1, \dots, k$. If $d_j = 1$, then $r_j(0) = b_j(0) - f(0)$, so again we have $\tilde{F}_j^l = [c_l r_j(0)]F_l' = [c_l b_j(0) - c_l f(0)][c_l f(0)]F_l = [c_l b_j(0)]F_l = \hat{F}_j^l$, for $l = 1, \dots, k$. Thus both \mathcal{C}_0 are equal and the verifier will accept.

Soundness. Let $I \subseteq \{0, 1, \dots, n\}$ with $|I| > t$. Given two accepting transcripts with different challenges (e.g. $d_j = 0$ and $d'_j = 1$, without loss of generality), if $0 \in I$ and any of $[c_1 r_j(0)]F_1 \neq [c_1 r'_j(0)]F_1'$, $[c_2 r_j(0)]F_2 \neq$

⁸ Regarding security assumptions, the fact that $f(0)$ cannot be obtained from $(F_1, F'_1 = [x]F_1)$ relies on the GAIP, while we additionally rely on \mathcal{C}_k -Vectorization Problem with Auxiliary Inputs (\mathcal{C}_k -VPwAI) to ensure that $f(0)$ cannot be obtained from the structured public key $(\mathcal{C}_k, F_1, F'_1, \dots, F_k, F'_k)$.

$[c_2 r'_j(0)]F'_2, \dots, [c_k r'_j(0)]F'_k \neq [c_k r'_j(0)]F'_k$, then we found a collision in \mathcal{C} . Similarly, if for some non-zero $i \in I$ we have $r_j(i) \neq r'_j(i) + x_i$ then we also have a collision for \mathcal{C} . If there is no collision, then

$$r_j(i) = r'_j(i) + x_i \text{ for all } i \in I, i > 0, \text{ and} \\ [c_l r_j(0)]F_l = [c_l r'_j(0)]F'_l \text{ for } l = 1, 2, \dots, k \text{ (if } 0 \in I),$$

so we can extract a valid witness as $r_j(X) - r'_j(X)$. \square

B Computational and Communication Costs of Our Protocols

In this section, we establish the computational and communication costs of our DKG protocols. We express the *sequential* costs τ of the protocol steps, i.e. the total runtime from start to finish, including when some of the parties are idle and discuss optimizations that minimize these idle times. We denote by T_I , T_E , T_C and T_H the cost of isogeny computations, polynomial evaluations, calls to the commitment scheme and calls to the random oracle, respectively. We ignore the cost of other operations, such as sampling and addition and multiplication over the ring \mathbb{Z}_N , as they are negligible in comparison. We express the communication cost in terms of *outgoing* communication cost γ per party. Let C_E and C_N denote the information content of an elliptic curve in \mathcal{E} and an element in \mathbb{Z}_N , respectively. A univariate polynomial of degree t can be represented by $t + 1$ elements in \mathbb{Z}_N . We first determine the costs of the individual building blocks of our protocol, before we put them together and compute the full costs.

VSS. We can easily see that in the VSS step from Fig. 1, each party first evaluates and sends out $2(n - 1)$ univariate polynomials. Then, in the verification step, parties further evaluate and share $2(n - 1)(n - 2)$ polynomial evaluations. We note that the evaluations can be done in parallel, thus this amounts to a total of $2(n - 1)^2$ sequential evaluations and $2(n - 1)(n + t - 1)$ elements in \mathbb{Z}_N sent out to the other parties. We find

$$\tau_{vss}(n) = 2(n - 1)^2 T_E \quad \text{and} \quad \gamma_{vss}(n, t) = 2(n - 1)(n + t - 1) C_N.$$

Proof Step. In the public key computation step of Fig. 1, parties have to compute one isogeny and run the proof in Algorithm 1. By carefully counting the operations in the latter, we find the total cost of

$$\tau_{proof}(n, \lambda) = \lambda(n + 1)T_E + (\lambda + 1)T_I + 2(n + 1)T_C + T_H.$$

After this step, the party has to publish the computed curve and the main proof and send the individual proof pieces to each other player. We can easily check that the proof pieces are 2λ bits each and that the main proof consists of $2(n + 1)$ commitments, each for 2λ bits and of the response, for $\lambda(t + 1)C_N$.

We note that both the computational and communication cost change when we use the twist trick. Remember that in this case, the challenge space increases from size 2 to 3, resulting in the number of repetitions being reduced to $\lambda' := \lceil \lambda / \log_2 3 \rceil$. In this case, the proof simply cost becomes $\tau_{proof}(n, \lambda')$. Regarding communication, we point out that the size of the proof pieces, determined by the security parameter λ , does not change when using the twist trick. To avoid confusion, we simply denote the cost of a commitment, or of a proof piece as $C_C = 2\lambda$, which is fixed. We can then express the total communication cost in the proof step as

$$\gamma_{proof}(n, t, \lambda) = C_E + (3n + 2)C_C + \lambda(t + 1)C_N .$$

Verification Step. For simplicity, we look at the upper bound $|Q| = n$. The verification step is reduced to the evaluation of Algorithm 2 by $n - 1$ parties, in parallel, once for $i = 0$ and once for $i \neq 0$. Note that the hash computation remains the same in both cases, and so only has to be computed once. By counting the different steps, we find the total of

$$\tau_{verif}(\lambda) = \lambda(T_E + T_I) + 4T_C + T_H .$$

If all the checks succeed, parties do not have to communicate anything in this step. In the converse case, per failed verification, parties have to broadcast one polynomial and verify at most n by evaluating them. This happens at most t times. We will ignore these costs in the interest of more realistic estimates.

Basic DKG Protocol. We can finally compute the full cost of the protocol in Fig. 1. This protocol simply consists of a VSS, and n consecutive proof and verification steps in the round-robin. We note that in the first round, we can use the twist trick. We find

$$\begin{aligned} \tau_{DKG}(n, \lambda) &= \tau_{vss}(n) + \tau_{proof}(n, \lambda') + \tau_{verif}(\lambda') \\ &\quad + (n - 1)(\tau_{proof}(n, \lambda) + \tau_{verif}(\lambda)) . \end{aligned}$$

and $\gamma_{DKG}(n, t, \lambda) = \gamma_{vss}(n, t) + \gamma_{proof}(n, t, \lambda') + (n - 1)\gamma_{proof}(n, t, \lambda)$. By looking at the individual terms, we find the results summarized in Tables 2 and 3.

Extended DKG Protocol. In the case of extended (non-structured) public keys discussed at the end of Sect. 3.2, the VSS step has to be repeated k times and the cost of a round-robin step naively increases by a factor k . This cost can be greatly improved by *staggering* the proofs and verifications, as was proposed in [17] and analyzed in more detail in [2]. Roughly, the idea is to compute the first proof and then publish it, so that other parties can verify it during the creation of the second proof and so on. As a result, the sequential cost of a round-robin step is reduced to the cost of k consecutive proofs plus one extra verification. But we can even do better, using the idea from [2, Sec. 6]: Since all

the different round-robins are independent computations, we can permute the players for each of them, and run multiple round-robins in parallel. This means, that while P_1 computes the k first curves for one secret and creates the PVP, P_2 does the same but for a different secret etc. Then, all of the verifications are performed, before moving onto the second step of all of the round-robins. In that way, we minimize idle time.

For n players with k secrets, the lowest attainable sequential runtime in this way is composed of $\lceil \frac{k}{n} \rceil$ proof steps and $k - \lfloor \frac{k}{n} \rfloor$ sequential verification steps, per round-robin step. Including the twist trick, we find the total cost

$$\begin{aligned} \tau_{DKG}^{ext.}(n, k, \lambda) &= k\tau_{vss}(n) + \left(\lceil \frac{k}{n} \rceil \tau_{proof}(n, \lambda') + \left(k - \lfloor \frac{k}{n} \rfloor \right) \tau_{verif}(\lambda') \right) \\ &\quad + (n - 1) \left(\lceil \frac{k}{n} \rceil \tau_{proof}(n, \lambda) + \left(k - \lfloor \frac{k}{n} \rfloor \right) \tau_{verif}(\lambda) \right). \end{aligned} \quad (3)$$

The communication costs are not changed by changing the order, so that we simply find $\gamma_{DKG}^{ext.}(n, k, t, \lambda) = k\gamma_{DKG}(n, t, \lambda)$. The individual terms are again summarized in Tables 2 and 3.

Structured DKG Protocol. If we use the DKG for structured public keys (given in Fig. 2), the VSS does not have to be repeated k times as we only have a single secret. Furthermore, in the public key computation step, proofs and verifications are done with SPVPs, which are introduced in Algorithms 3 and 4. Some scrutiny reveals

$$\begin{aligned} \tau_{proof}^{SPVP}(n, k, \lambda) &= \lambda(n + 1)T_E + k(\lambda + 1)T_I + 2(n + 1)T_C + T_H, \\ \tau_{verif}^{SPVP}(k, \lambda) &= \lambda T_E + k\lambda T_I + 4T_C + T_H. \end{aligned}$$

Note that τ_{proof}^{SPVP} also includes the computation of the curves in the round-robin step. In comparison to the cost of the standard PVPs established earlier, only the isogeny computations increase by a factor k , while the other terms remain unchanged. Regarding communication cost, we can easily see that an SPVP has the same size as a PVP, independent of k . The difference to the basic case is that we publish k curves instead of one, resulting in the cost per proof step of $\gamma_{proof}^{SPVP}(n, k, t, \lambda) = kC_E + (3n + 2)C_C + \lambda(t + 1)C_N$. We end up with the total

$$\gamma_{DKG}^{SPK}(n, k, t, \lambda) = \gamma_{vss}(n, t) + \gamma_{proof}^{SPVP}(n, t, \lambda') + (n - 1)\gamma_{proof}^{SPVP}(n, t, \lambda).$$

In the protocol from Fig. 2, we can use a similar approach as for the extended DKG protocol, in the sense that we can run multiple round-robins in parallel. A difference here, is that each player does not run k individual PVPs, but instead batches them into SPVPs. This allows to run an initial round of n SPVPs in parallel, each with $\lfloor \frac{k}{n} \rfloor$ elements, and a second round with $k \bmod n$ PVPs in parallel. The first round has $n - 1$ subsequent verifications to be performed and the second $k \bmod n$ more, again all in parallel by the individual players. The cost per round-robin step can therefore be expressed as

$$\begin{aligned} R(n, k, \lambda) &= \tau_{proof}^{SPVP}(n, \lfloor \frac{k}{n} \rfloor, \lambda) + (n - 1)\tau_{verif}^{SPVP}(\lfloor \frac{k}{n} \rfloor, \lambda) \\ &\quad + \chi_{n,k}(\tau_{proof}(n, \lambda) + (k \bmod n)\tau_{verif}(\lambda)), \end{aligned}$$

where we define $\chi_{n,k} = \lceil \frac{k}{n} \rceil - \lfloor \frac{k}{n} \rfloor$, i.e. $\chi_{n,k} = 0$, if $n \mid k$, and 1 otherwise. These steps are repeated n times, where at the first step we can use the twist trick. Together with the VSS, we find the total cost of

$$\tau_{DKG}^{str.}(n, k, \lambda) = \tau_{vss}(n) + R(n, k, \lambda') + (n - 1)R(n, k, \lambda). \tag{4}$$

Again, the individual terms are summarized in Tables 2 and 3.

Comparison of Extended and Structured Case. Finally, we establish some of the background related to Figs. 4 and 3.

Communication. Using the fact that $N \approx \sqrt{p}$ [29] and choosing the security parameter $\lambda \approx \sqrt[4]{p}$ (reflecting the classical security, see [2, 10]), we can easily identify $2\lambda \approx C_C \approx C_N \approx \frac{1}{2}C_E$. By plugging this into the terms in Table 3 and dropping some of the constant terms, we can see, that the communication cost of the extended DKG asymptotically scales with $2nk\lambda(5n + \lambda t)$, while the structured case scales with $2n\lambda(5n + \lambda t + 2k)$. For $n \rightarrow \infty$, the latter is k times smaller, while for $k \rightarrow \infty$, the latter is $\lambda t/2$ times smaller, both considerable gains. We depict these trends in Fig. 4. The asymptotic quadratic trend in n and linear trend in k of our schemes are clearly visible in the figure. One can also see, that the expected asymptotic gain of $k = 2^6$ for the structured case with respect to the extended case is well-represented on the left graph, while the right graph shows the expected asymptotic gain of $\lambda n/6 \approx 64$.

Computation. The results in Table 2 show that using structured public keys removes the dependency on k in all cases but isogeny computations. It is clear that the number of calls to commitment schemes or random oracles becomes the same around $k \approx n$. For the number of polynomial evaluations, this behavior becomes a bit more complex, and the structured case always outperforms the extended case for some $k \leq n$. This is due to the fact that the VSS in the extended case scales with k , while it is independent of k in the structured case.

We note that in general, isogeny computation costs will strongly dominate the full protocol cost. We restate the full isogeny costs of both protocols here, in the most general case, using the twist trick. For the latter, we define $\lambda' = \lceil \lambda / \log_2 3 \rceil$. By looking at the isogeny cost terms of Eqs. (3) and (4), we find, after some arithmetic, that they are both equal to

$$I(n, k, \lambda) = (\lambda' + (n - 1)\lambda)(k + \chi_{n,k}) + n \lceil \frac{k}{n} \rceil .$$

We compare this with the results from [2] in Fig. 3. Below, we also summarize the gains we get by using the twist trick for low n .

n	2	3	4	5	6	8	10	20	50
Gain	18.3%	12.2%	9.2%	7.3%	6.1%	4.6%	3.6%	1.8%	0.7%




References

1. Asharov, G., Lindell, Y.: A full proof of the BGW protocol for perfectly secure multiparty computation. *J. Cryptology* **30**(1), 58–151 (2017)
2. Atapoor, S., Bagheri, K., Cozzo, D., Pedersen, R.: CSI-SharK: CSI-FiSh with sharing-friendly keys. *Cryptology ePrint Archive*, Report 2022/1189 (2022). <https://eprint.iacr.org/2022/1189>
3. Bagheri, K., Cozzo, D., Pedersen, R.: An isogeny-based ID protocol using structured public keys. In: Paterson, M.B. (ed.) *IMACC 2021*. LNCS, vol. 13129, pp. 179–197. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92641-0_9
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press (1988)
5. Bernstein, D., De Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. *arXiv preprint arXiv:2003.10118* (2020)
6. Beullens, W., Disson, L., Pedersen, R., Vercauteren, F.: CSI-RAShI: distributed key generation for CSIDH. In: Cheon, J.H., Tillich, J.-P. (eds.) *PQCrypto 2021*. LNCS, vol. 12841, pp. 257–276. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81293-5_14
7. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) *ASIACRYPT 2019*. LNCS, vol. 11921, pp. 227–247. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_9
8. Campos, F., Muth, P.: On actively secure fine-grained access structures from isogeny assumptions. In: Cheon, J.H., Johansson, T. (ed.) *PQCrypto 2022*. LNCS, vol. 13512, pp. 375–398. Springer (2022)
9. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH (preliminary version). *Cryptology ePrint Archive*, Report 2022/975 (2022). <https://eprint.iacr.org/2022/975>
10. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) *ASIACRYPT 2018*. LNCS, vol. 11274, pp. 395–427. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_15
11. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. *J. Cryptology* **22**(1), 93–113 (2009)
12. Jean Marc Couveignes: Hard homogeneous spaces. *IACR Cryptol. ePrint Arch.* **2006**, 291 (2006)
13. Cozzo, D., Smart, N.P.: Sashimi: cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In: Ding, J., Tillich, J.-P. (eds.) *PQCrypto 2020*. LNCS, vol. 12100, pp. 169–186. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_10
14. De Feo, L.: Mathematics of isogeny based cryptography. *arXiv preprint arXiv:1711.04062* (2017)
15. De Feo, L., Jao, D., Plüt, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Cryptology ePrint Archive*, Report 2011/506 (2011). <https://eprint.iacr.org/2011/506>
16. De Feo, L., Kohel, D., Leroux, A., Petit, C., Wesolowski, B.: SQISign: compact post-quantum signatures from quaternions and isogenies. In: Moriai, S., Wang, H. (eds.) *ASIACRYPT 2020*. LNCS, vol. 12491, pp. 64–93. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_3

17. De Feo, L., Meyer, M.: Threshold schemes from isogeny assumptions. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12111, pp. 187–212. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45388-6_7
18. De Feo, L., et al.: SCALLOP: scaling the csi-fish. IACR Cryptol. ePrint Arch., p. 58 (2023)
19. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust and efficient sharing of RSA functions. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 157–172. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_13
20. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology* **20**(1), 51–83 (2007)
21. Kitaev, A.Y.: Quantum measurements and the abelian stabilizer problem. *Electron. Colloquium Comput. Complex.*, TR96-003 (1996)
22. Maino, L., Martindale, C.: An attack on SIDH with arbitrary starting curve. *Cryptology ePrint Archive, Report 2022/1026* (2022). <https://eprint.iacr.org/2022/1026>
23. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 522–526. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_47
24. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
25. Robert, D.: Breaking SIDH in polynomial time. *Cryptology ePrint Archive, Report 2022/1038* (2022). <https://eprint.iacr.org/2022/1038>
26. Rostovtsev, A., Stolunov, A.: Public-key cryptosystem based on isogenies. *IACR Cryptol. ePrint Arch.* **2006**, 145 (2006)
27. Shamir, A.: How to share a secret. *Commun. ACM* **22**(11), 612–613 (1979)
28. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134 (1994)
29. Siegel, C.: Über die classenzahl quadratischer zahlkörper. *Acta Arithmetica* **1**(1), 83–86 (1935)
30. Silverman, J.H.: *The arithmetic of elliptic curves*, vol. 106. Springer Science & Business Media (2009)
31. Stinson, D.R., Wei, R.: Unconditionally secure proactive secret sharing scheme with combinatorial structures. In: Heys, H., Adams, C. (eds.) SAC 1999. LNCS, vol. 1758, pp. 200–214. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46513-8_15
32. Stolunov, A.: *Cryptographic schemes based on isogenies* (2012)
33. Unruh, D.: Computationally binding quantum commitments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 497–527. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_18
34. Vélu, J.: Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris, Séries A*, **273**, 305–347 (1971)
35. Wu, Q., Chen, H., Li, Z., Jia, C.: On a practical distributed key generation scheme based on bivariate polynomials. In: *2011 7th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–4 (2011)



Efficient Isogeny Proofs Using Generic Techniques

Kelong Cong¹(✉) , Yi-Fu Lai²(✉) , and Shai Levin²(✉) 

¹ imec-COSIC, KU Leuven, Leuven, Belgium
kelong.cong@esat.kuleuven.be

² University of Auckland, Auckland, New Zealand
ylai276@aucklanduni.ac.nz, shai.levin@auckland.ac.nz

Abstract. Generating supersingular elliptic curves of unknown endomorphism ring has been a problem vexing isogeny-based cryptographers for several years. A recent development has proposed a trusted setup protocol to generate such a curve, where each participant generates and proves knowledge of an isogeny. Thus, the construction of efficient proofs of knowledge of isogeny has developed new interest.

Historically, the isogeny community has assumed that obtaining isogeny proofs of knowledge from generic proof systems, such as zkSNARKs, was not a practical approach. We contribute the first concrete result in this area by applying Aurora (EUROCRYPT'19), Ligerio (CCS'17) and Limbo (CCS'21) to an isogeny path relation, and comparing their performance to a state-of-the-art, tailor-made protocol for the same relation. In doing so, we show that modern generic proof systems are competitive when applied to isogeny assumptions, and provide an order of magnitude (3-10×) improvement to proof and verification times, with similar proof sizes. In addition, these proofs provide a stronger notion of soundness, and statistical zero-knowledge; a property that has only recently been achieved in isogeny PoKs. Independently, this technique shows promise as a component in the design of future isogeny-based or other post-quantum protocols.

Keywords: Isogeny · Zero-knowledge · zkSNARK · Interactive Oracle Proof · MPC-in-the-Head

1 Introduction

Isogeny-based cryptography was first introduced with the CGL hash function [24] by Charles, Goren and Lauter, where the core hardness assumption is that, given two isogenous elliptic curves, it is hard to recover an isogeny between them. Several other isogeny-based protocols were proposed, including SIDH [48], which relaxes the assumption by giving additional torsion point information; CSIDH based on group actions [23, 54]; SQI-Sign, a signature scheme based on endomorphism rings [31]; and pSIDH, a relative of SIDH which uses a different isogeny representation [51]. Even though there was a recent cryptanalysis breakthrough on SIDH [22, 52, 56], other cryptosystems (not based on SIDH) remain unaffected,

such as [23, 31, 51]. Additionally, a variety of advanced schemes and protocols based on isogenies, such as oblivious transfer and exotic signatures, have been proposed in the literature [13–15, 21, 50].

In every isogeny-based cryptosystem, isogeny walks start from a public curve. In the literature, the candidate is usually one of the j -invariants 0 or 1728 with a known endomorphism ring. In isogeny-based constructions, sampling an elliptic curve without knowing its endomorphism ring [18, 53], is currently a computationally infeasible task, and is essential in some constructions and applications [1, 21, 24, 50, 58]. From a cryptanalytical perspective, having a public curve with an unknown endomorphism ring significantly reduces the information an attacker/analyst may have. A recent proposal [6] suggests a trusted setup ceremony to resolve this problem. In the ceremony, every party computes an isogeny path from the previous curve to another, produces a proof that the isogeny was generated honestly, and disposes of the path. They then publish their new curve and associated proof publicly, which all parties verify. Once every participant has completed their round, the ceremony outputs the final curve. As long as at least one party behaves honestly, recovering the final curve’s endomorphism ring is difficult, even if the rest of the participants collude.

However, generating a zero-knowledge proof of an isogeny path is not a trivial task in general. In the realm of group actions, it is not difficult to achieve and the proofs for more sophisticated relations can be made [4, 13–15]. However, out of realm of the group actions, the task has been known to be difficult to achieve either soundness (for the exact relation) or (statistical) zero-knowledge, with some protocols requiring ad-hoc security assumptions. The state-of-the-art line of work is given in [6, 27, 30, 42], yet there is still room for improvement. Suppose 300 participants run the ceremony single-threaded on a normal machine, the protocol will take roughly an hour to complete for $\lambda = 128$, and 13 h for $\lambda = 256$.

Historically, it was assumed that tailor-made proof systems for isogeny relations performed better than generic ones. However, the developments of generic proof systems, such as zkSNARKs¹, which allow a prover to prove or argue the knowledge of any NP relation, have advanced the field significantly in recent years. zkSNARKs enable a prover to produce a publicly-verifiable proof in a zero-knowledge and non-interactive manner. Moreover, the proof size is *succinct*, sublinear in the size of the witness, and the verification time is much shorter than producing the proof. The area of zero-knowledge proof systems has been very active [3, 10, 17, 19, 33, 47, 49] (see [45, 59] for surveys). These generic proof systems work well with symmetric primitives and have applications in post-quantum cryptosystems [7, 32, 36, 37, 41, 63], and privacy-preserving blockchain protocols such as [9].

Applying generic proof systems to isogeny-based cryptography remains uncommon. Though there exists a verifiable delay function from isogenies using a SNARG² [25], it is not in zero-knowledge, and the result remains theoretical in nature, with unclear practicality. In particular, due to the complexity of

¹ Zero-knowledge, succinct, non-interactive, arguments of knowledge.

² Succinct, non-interactive argument.

computing isogenies, size and the structure of the operating field, using generic proof systems in isogeny-based cryptography appears challenging and impractical. Generic proof systems have been applied to protocols utilising fields of bit length at most 256-bits, whereas many isogeny-based protocols utilise field extensions of a field of upwards of 512-bits. Due to these factors, it was previously assumed these proof systems did not scale well with isogeny-based protocols. In the isogeny community, the plausibility of the following question was largely disputed:

Can generic proof systems serve as a practical tool in isogeny-based cryptography?

1.1 Contribution

We affirm the question above. That is, generic proof systems are remarkably efficient for isogeny-based cryptography. Specifically, our contributions are:

- We propose a non-interactive protocol to prove knowledge of an isogeny path using a generic zkSNARK proof system for R1CS (rank-1 constraint systems). We achieve this by re-writing the isogeny path relation into a compact R1CS representation and then applying existing (plausibly) post-quantum proof systems [3, 10, 33]. The PoK inherits the properties of soundness and statistical zero-knowledge from the underlying proof systems, and supports supersingular isogeny graphs operating over any cryptographically sized prime of the form $p = 2^a f' \pm 1$ for every $f' \in \mathbb{N}$ with isogeny paths of arbitrary length.
- We provide an alternative set of parameters of the form $p = 2^a 3^b f + 1$ with equivalent security to those from SIKE to aid in our testing. These parameters are designed to better support the requirements of the underlying proof systems.
- Our protocol is implemented as a proof of concept, and we report benchmark results for a variety of parameters. Using our R1CS instances from above, the generic proof systems yield competitive results as isogeny identification schemes. In particular, by utilising Aurora [10] our proof systems are roughly 3–10 times faster than the state-of-the-art [6] of the same walk length, while maintaining a similar proof size.

1.2 Related Work

The motivation behind this work is to construct an efficient isogeny proof of knowledge. One such application of which is a multi-party setup protocol to generate a supersingular curve of unknown endomorphism ring, introduced in [6]. We give a brief history of prior works on proving isogeny knowledge, which differs from our approach.

Prior to this work, isogeny proofs of knowledge have existed in different forms, notably [27, 29]. These are Σ -protocols, tailored to the specific nature of isogeny computation, and follow a direction to the original DJP identification protocol. These protocols can be viewed as revealing different edges on the SIDH square (Fig. 1) in order to prove knowledge of the isogeny $\phi : E_0 \rightarrow E_1$ of degree $\ell_A^{e_A}$. To generate the square, the prover computes an isogeny $\psi : E_0 \rightarrow E_2$ of degree $\ell_B^{e_B}$. The prover then determines the isogenies ϕ' and ψ' by their kernels, such that $\ker \phi' = \psi(\ker \phi)$ and $\ker \psi' = \phi(\ker \psi)$.

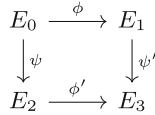


Fig. 1. The SIDH square

In the original De Feo-Jao-Plüt identification protocol, in each iteration of the Σ -protocol, the prover generates a new SIDH square in the manner described above (with a fresh choice of ψ). The prover reveals the curves E_2, E_3 . The verifier then sends a binary challenge b . If $b = 0$, the prover sends the vertical isogenies to the verifier, who checks if they are indeed isogenies of correct degree, domain, and codomain. Likewise, if $b = 1$, the prover sends the horizontal isogeny ϕ' , and the prover verifies the isogeny is of correct degree and domain/codomain.

However, this protocol suffered from various issues. Aside from an ad-hoc security assumption, it did not achieve statistical zero-knowledge (since the side ϕ' is strongly correlated to the side ϕ), and possessed issues with its proof of soundness (see [27, 43]). The former work increases the challenge space to 3, and proposed a solution to soundness by including a commitment to $\ell_B^{e_B}$ -torsion bases of E_2 and E_3 , such that the latter is the image of the former under ϕ' .

The latest work, SECUER PoK [6], resolves the problem of statistical zero-knowledge (and forgoing the need for additional assumptions) by extending the degree of ϕ and ψ by composing isogenies and gluing SIDH squares together such that the walk ψ causes uniform mixing in a particular lift of the supersingular isogeny graph³, causing the distribution of (E_1, ϕ') to be statistically close to uniform. In addition, also they extend the path length of the ϕ to guarantee the image curve of the isogeny is uniform. This means that in their setup protocol, provided a participant is honest, the output j -invariant is uniformly at random in the set of supersingular elliptic curves.

However, there are still some problems with these approaches. The increased challenge space of [6, 27] yields a knowledge error of $\frac{2}{3}$ per round, which increases the number of repetitions required to achieve a sufficient soundness level. Furthermore, SECUER PoK relies on a relaxed assumption to soundness, namely that

³ The supersingular isogeny graph with level d Borel structure, where $d = |\ker \phi|$.

the extractor may not obtain the original isogeny ϕ , but an isogeny $\phi' = [\ell_B^{2i}] \circ \phi$ for some $i \leq e_B$.

We forgo these approaches (and their soundness issues) by viewing the isogeny, ϕ , as a walk on the supersingular ℓ_A -isogeny graph and then proving the knowledge of the walk with a generic proof system. Provided the prover can efficiently compute the intermediate j -invariants on the walk, which is done in practice using Vélu's formulae, this provides the same functionality as the proof systems above.

2 Preliminaries

2.1 Notations

A function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ is negligible if for every polynomial p there is an N such that for all $n > N$ it holds that $f(n) < \frac{1}{p(n)}$. Given a relation \mathcal{R} , we say that $\mathcal{L}(\mathcal{R})$ is the set of all elements x such that there exists a w where $(x, w) \in \mathcal{R}$.

2.2 Isogeny Graphs

This section recalls a few essential properties of supersingular elliptic curves relevant for our work. We refer to [57, 61] for a more extensive exposition.

Elliptic Curves. An elliptic curve is a projective non-singular curve of genus one, with a marked point. If the coefficients of a curve are defined over a field K , we say the curve is defined over K . The K -rational points, $E(K)$, form a group under an additive operator. Elliptic curves over a field K may be uniquely identified (up to isomorphism) by a single field element in K , called the j -invariant. The j -invariant is efficiently computable given a curve's coefficients.

Isogenies. An isogeny is a surjective morphism of elliptic curves preserving both geometric structure (as a rational map) and group structure (as a group homomorphism). The degree of a separable isogeny is the size of its kernel as a group homomorphism. We say an isogeny is an ℓ -isogeny if it has degree ℓ , and that two elliptic curves are ℓ -isogenous if there exists an ℓ -isogeny between them. We shall assume all isogenies discussed in this work are separable (but need not necessarily be cyclic).

Supersingular ℓ -Isogeny Graph. We denote the supersingular ℓ -isogeny graph over \mathbb{F}_{p^2} as $G_\ell(p)$, whose vertices are the supersingular elliptic curves over the field (up to isomorphism), with an edge between two vertices if they are ℓ -isogenous⁴. It is a well known fact that for $\ell \neq p$, apart from the vertices $j = 0, 1728$; $G_\ell(p)$ is a Ramanujan graph [55], an optimal expander graph.

⁴ We view the edges as undirected by identifying each isogeny with its dual, and identify edges which are equivalent up to post-composition with an isomorphism.

Modular Polynomial. The modular polynomial $\Phi_\ell(X, Y)$, is a symmetric polynomial of degree $\ell + 1$ whose roots over \mathbb{F}_{p^2} correspond to every pair of ℓ -isogenous j -invariants of elliptic curves over \mathbb{F}_{p^2} . This allows us to efficiently determine if two elliptic curves are ℓ -isogenous over a given field. For $\ell = 2$, we have the modular polynomial

$$\begin{aligned} \Phi_2(X, Y) = & X^3 + Y^3 - 162000(X^2 + Y^2) + 1488XY(X + Y) - X^2Y^2 \\ & + 8748000000(X + Y) + 40773375XY - 15746400000000. \end{aligned} \quad (2)$$

So, two j -invariants j_1, j_2 are adjacent in $G_\ell(p)$ if and only if $\Phi_\ell(j_1, j_2) = 0 \pmod p$.

2.3 Proof Systems

Zero-Knowledge Succinct Non-interactive Arguments of Knowledge.

In the (explicitly programmable) random oracle model, a *zero-knowledge non-interactive succinct argument⁵ of knowledge* (zkSNARK) for a relation $\mathcal{R} = \{(x, w)\}$ is a tuple (P, V) where P, V are probabilistic polynomial time (PPT) algorithms with access to a random oracle ρ which satisfy the following properties:

- COMPLETENESS: For every $(x, w) \in \mathcal{R}$, $\lambda \in \mathbb{N}$,

$$\Pr[V^\rho(x, \pi) = 1 \mid \pi \leftarrow P^\rho(x, w)] = 1$$

- SOUNDNESS: Given negligible soundness s , for every PPT \tilde{P} , $x \notin \mathcal{L}(\mathcal{R})$, and $\lambda \in \mathbb{N}$:

$$\Pr[V^\rho(x, \pi) = 1 \mid \pi \leftarrow \tilde{P}^\rho(x)] \leq s(x, \lambda).$$

- PROOF OF KNOWLEDGE: Given negligible knowledge error κ , there exists a PPT extractor E such that, for every x , PPT \tilde{P} , $\lambda \in \mathbb{N}$,

$$\Pr[(x, w) \in \mathcal{R} \mid w \leftarrow E^{\tilde{P}}(x, 1^\lambda)] - \Pr[V^\rho(x, \pi) = 1 \mid \pi \leftarrow \tilde{P}^\rho] \leq \kappa(x, \lambda).$$

Where the extractor E may program the responses to random oracle queries of \tilde{P} , and either get a response of the next query or output π , at which point \tilde{P} goes to the start of its computation with the same randomness and auxiliary input.

- ZERO KNOWLEDGE: A non-interactive protocol (P, V) is statistical zero-knowledge (with negligible function z) in the explicitly programmable random oracle model (EPRO)⁶, if there exists a PPT simulator \mathcal{S} , such that for every $(x, w) \in \mathcal{R}$ and unbounded distinguisher D :

$$\Pr[D^{\rho^{[\mu]}}(\pi) = 1 \mid (\pi, \mu) \leftarrow \mathcal{S}^\rho(x)] - \Pr[D^\rho(\pi) = 1 \mid \pi \leftarrow P^\rho(x, w)] \leq z(x, \lambda),$$

⁵ Typically, a non-interactive random-oracle proof system is a *proof* (NIZKPoK) only if the definition of soundness holds given a computationally unbounded prover, and is otherwise called an *argument*. We may use the terms interchangeably to refer to both.

⁶ We include the definition of zero-knowledge in the EPRO model, which is required in the application of the BCS transform—the Fiat-Shamir analogue for IOPs.

where the EPRO, $\rho[\mu]$, outputs $\mu(x)$ if x is in the domain of μ , otherwise it outputs $\rho(x)$. The distributions are taken over the uniformly at random instantiation of ρ and the randomness of P, V .

- **SUCCINCTNESS:** A proof system (P, V) for a relation \mathcal{R} is *succinct*, if, for any $(x, w) \in \mathcal{R}$ and corresponding proof $\pi \leftarrow P^p(x, w)$, π grows polylogarithmically in w . In particular, $|\pi| = \text{poly}(\lambda, |x|, \log(|w|))$.

Interactive Oracle Proofs. An *interactive oracle protocol* between two PPT algorithms A and B over k rounds is a protocol where at the i th round, A sends an i -th message m_i to B , who responds with a random access oracle f_i which may be queried in consequent rounds. After k rounds, A either accepts or rejects (see [11] for details).

An *Interactive Oracle Proof* (P, V) for a relation \mathcal{R} with round complexity k and soundness s satisfies the following properties:

- **COMPLETENESS:** For every $(x, w) \in \mathcal{R}$, $(P(x, w), V(x))$ is a $k(x)$ -round interactive protocol with accepting probability 1.
- **SOUNDNESS:** For every $x \notin \mathcal{L}(\mathcal{R})$ and every \tilde{P} , $(\tilde{P}, V(x))$, is a $k(x)$ -round interactive oracle protocol with accepting probability at most $s(x)$.

Interactive Oracle Proofs (IOPs), introduced by Ben-Sasson et al [11], are a generalisation of both Interactive Proofs (IPs) and Probabilistically Checkable Proofs (PCPs). One may note that IOPs directly generalise PCPs to multiple rounds. The motivation behind the construction of IOPs is that of efficiency, by minimising redundancy that might be present in a traditional 1 round PCP construction. Analogously to IPs and PCPs, an IOP may also satisfy the properties of zero-knowledge, proof of knowledge, and succinctness, as well as a transformation which performs similarly to the Fiat-Shamir transform [38]. Thus, zkSNARKs can be obtained from IOPs. Intuitively, succinct proofs are achievable when the prover sends random access oracles (instantiated via Merkle trees with a CRH function), rather than full length messages.

Theorem 1 (BCS Transform). *There exists a transform T that inputs an IOP (P, V) and outputs a non-interactive argument of knowledge (P^*, V^*) that preserves proof of knowledge and succinctness. Moreover, when the underlying IOP is statistically zero-knowledge, the resulting protocol is statistically zero-knowledge under the EPRO model.⁷*

Proof. See [11, Sec. 6]

In this work, we consider IOPs that satisfy all of these properties and are also *transparent*. That is, secure in the absence of the common reference string (CRS) model, in which protocols require trusted setup.

⁷ In particular, the extractor in the transformation T is straight-line, and does not apply the forking lemma.

2.4 Rank-1 Constraint Systems

We recall the definition of rank-1 constraint systems (R1CS), which some zkSNARKs (e.g., Aurora) take as an input. R1CS is parameterized by $n, m \in \mathbb{N}$ and a prime power q , and consists of instance-witness pairs $((A, B, C, v), w)$ where $A, B, C \in \mathbb{F}_q^{m \times (n+1)}$ and v, w are vectors over \mathbb{F}_q such that

$$Az \circ Bz = Cz$$

for $z := (1, v, w) \in \mathbb{F}_q^{n+1}$, where \circ denotes coordinate-wise (Hadamard) product. Conceptually, A, B, C encode constraints on variables v, w ; where v contains (public) auxiliary input, and w contains both secret input and intermediate variables in a computation. R1CS may encode arithmetic circuit satisfiability. In Sect. 3.4, we encode level ℓ modular polynomials into an R1CS instance, along with some optimisations, to prove knowledge of an isogeny path.

2.5 MPC-in-the-Head

The MPC-in-the-Head (MPCitH) paradigm was introduced in the seminal work of Ishai et al. [46] Suppose the prover wishes to convince a verifier of an NP relation \mathcal{R} in zero-knowledge, where x is the instance and w is the witness. The prover simulates an MPC protocol with n parties locally (in its head) and commits to the transcript. The verifier asks the prover to decommit a subset of the transcript and check whether the messages are consistent and that the reconstructed output is 1, meaning that the relation \mathcal{R} holds. If there are no failures during the verification, the verifier accepts the proof. Intuitively, completeness holds trivially, (statistical) zero-knowledge holds if the decommitted transcript is not enough to reveal the full transcript (e.g., revealing $n - 1$ transcripts reveals nothing about the full transcript when using additive secret sharing). Regarding soundness, the prover may cheat if the faulty transcript is not challenged by the verifier. Nevertheless, it is possible to boost the soundness by repeating the protocol many times. Using the Fiat-Shamir transform [38] it is possible to convert an interactive protocol to a non-interactive one.

Limbo. Limbo [33] is the state-of-the-art non-interactive zero-knowledge proof of knowledge for arithmetic circuit satisfiability protocol based on the MPCitH paradigm. Despite not satisfying the asymptotic definition of succinctness, Limbo has proven to have good concrete efficiency for small to medium sized circuits (i.e. circuits with less than 500000 multiplication gates). Thus we include it in our consideration. For the detailed description, we refer the reader to the paper.

2.6 Reed-Solomon IOPs

The other line of protocols [3, 10, 17] we consider in this work is called Reed-Solomon IOPs. In contrast to the MPCitH-based approach above, these protocol

achieve better asymptotic performance, with proof sizes scaling either sublinearly or polylogarithmically in witness length (i.e. succinct).

At a high level, in RS-IOPs, the witness w corresponds to the input plus all the intermediate variables in the computation. The prover transforms the witness w into various vectors, depending on the proof system, which are then encoded with a RS code. The verifier engages in various sub-protocols with the prover to check conditions on the RS encoded values to convince itself that the encoded values form valid RS codewords and satisfies the constraints given in the relation.

Reed-Solomon Codes. Given an ordered subset $L = \{\ell_1, \dots, \ell_k\}$ of a field \mathbb{F}_q and $\alpha \in (0, 1]$, we denote $RS[L, \alpha] \subseteq \mathbb{F}_q^k$ to be the set of evaluations over L of all polynomials of degree less than αk . That is, a codeword c is in $RS[L, \alpha]$ if and only if there exists a polynomial p of degree less than αk such that the $c = (p(\ell_1), \dots, p(\ell_k))$.

Aurora. Aurora is a transparent zkSNARK for the R1CS relation secure in the EPRO. At a high level, Aurora's underlying IOP reduces to proving the following two subproblems:

- ROWCHECK: Given vectors $a, b, c \in \mathbb{F}_q^m$, test whether $a \circ b = c$
- LINCHECK: Given vectors $x \in \mathbb{F}_q^m$, $y \in \mathbb{F}_q^{n+1}$, and matrix $M \in \mathbb{F}_q^{m \times (n+1)}$; test whether $x = My$.

Given IOPs for these problems, one may construct an IOP for R1CS. Given an R1CS instance $((q, n, m, A, B, C, v), w)$, the prover sends four oracles to the verifier: the satisfying assignment for z , $y_A := Az$, $y_B := Bz$, and $y_C := Cz$. The prover then engages in parallel execution of the following:

- An IOP for ROWCHECK to verify that $y_A \circ y_B = y_C$.
- An IOP for LINCHECK to verify that $y_A = Az$, $y_B = Bz$, and $y_C = Cz$.

Finally, the verifier checks that z is consistent with the auxiliary input v .

However, such a protocol would be neither succinct, nor zero-knowledge. In order for the protocol to achieve sublinear communication complexity, the subprotocols for LINCHECK and ROWCHECK both utilise Reed-Solomon encoded variants. In this case, foregoing zero-knowledge, the subroutines for LINCHECK and ROWCHECK encode the vectors y_A, y_B, y_C as the coefficients of a unique polynomial that matches them over some $H_1 \subset \mathbb{F}_q$ where $|H_1| = m$, and likewise for z , as the coefficients of a polynomial that matches z over some $H_2 \subseteq \mathbb{F}_q$ where $|H_2| = n + 1$. In addition, some extra work is done to check the degree of the polynomials is consistent with the input via a *low-degree* test. Aurora utilises the FRI protocol [8] to achieve this efficiently.

Zero knowledge is achieved by encoding a vectors Az, Bz, Cz not as unique polynomial of degree $|H_1| - 1$ matching the entries of Az, Bz, Cz on H_1 , but as a random polynomial of degree $|H_1| + m$ conditioned on matching Az, Bz, Cz on H_1

(the same process applies to z with domain H_2). The polynomial is represented as evaluations over a domain L disjoint from H_1 and H_2 such that m queries cannot leak any information about v . In order to guarantee these subsets are disjoint, over a prime field, the subsets H_1, H_2 are chosen to be multiplicative subgroups of the field (of order a power of two such that $H_1 \subseteq H_2$ or $H_2 \subseteq H_1$), and the evaluation domain L is a multiplicative coset of a subgroup of $H_1 \cup H_2$.

Ligero. Ligero [3] is another generic proof system based on an RS-IOP for boolean or arithmetic circuit satisfiability (technically, an IPCP, as it only comprises of a single round). Ligero satisfies all of the properties of a zkSNARK except for succinctness. Rather, it is *sublinear*, with proof length scaling in square root of the circuit size. Given an arithmetic circuit C of N gates, a Ligero prover represents the satisfying assignment of the s ($\approx N$) wires of C into a slightly redundant matrix representation of size $O(\sqrt{s}) \times O(\sqrt{s})$, and encodes each row of this matrix using an (interleaved) RS code. The verifier challenges the prover to reveal linear combinations of the entries of the codeword matrix, which is checked against λ randomly selected columns of the matrix which are consequently revealed by the prover.

Aside from the underlying proof relation, the key distinction between Aurora and Ligero is informed by two design decisions: Ligero encodes its oracles with $O(\sqrt{N})$ RS codewords of length $O(\sqrt{N})$, rather than by a single RS codeword of length $O(N)$. In addition, it uses a direct (single-round) low-degree test rather than the FRI IOP.

Remark 1. Due to the closed-source implementation of Ligero, we apply our testing with a modified variant, *RICS-Ligero* [10, App. B] which supports RICS.

3 Construction

3.1 Hardness Assumptions and Relations

Recent attacks have rendered the SIDH assumption broken [22, 52, 56]. The key insight is that these attacks require the image of the torsion points P_1, Q_1 , however, the following, more general isogeny path-finding problem below, historically used to cryptanalyse SIDH, remains unaffected.

Problem 1 (ISOPATH). Given supersingular elliptic curves E_0, E_1 defined over \mathbb{F}_{p^2} , find an isogeny $\phi : E_0 \rightarrow E_1$ such that $\deg \phi = \ell^k$ for a fixed prime ℓ and some $k \in \mathbb{Z}$.

We define the following relation based on the hardness of ISOPATH:

$$\mathcal{R}_{\ell^k\text{-ISOPATH}} = \{((E_0, E_1), \phi) : \phi : E_0 \rightarrow E_1 \text{ is an isogeny, } \deg \phi = \ell^k, k \in \mathbb{Z}\}$$

The isogeny witness ϕ is typically represented by fixing a basis of the ℓ^k -torsion group, and giving a kernel generator, a point on E_0 of order ℓ^k . Instead, we choose to represent our witness isogeny ϕ in the relation above by using the

modular polynomial. Recall, two elliptic curves E, E' are ℓ -isogenous if and only if $\Phi_\ell(j(E), j(E')) = 0$. Then an isogeny $\phi : E_0 \rightarrow E_1$ of degree ℓ^k can equivalently be represented as a sequence of intermediate j -invariants j_1, j_2, \dots, j_{k-1} such that

$$\begin{aligned} \Phi_\ell(j(E_0), j_1) &= 0 \\ \Phi_\ell(j_i, j_{i+1}) &= 0 \quad \text{for all } i \in \{1, \dots, k-2\} \\ \Phi_\ell(j_{k-1}, j(E_1)) &= 0 \end{aligned}$$

Hence, more precisely, the relation we prove is as follows:

$$\mathcal{R}_{\ell^k\text{-MODPOLY}} = \left\{ ((E_0, E_1), (j_i)_{i \in \{1, \dots, k-1\}}) : \begin{array}{l} \Phi_\ell(j(E_0), j_1) = 0, \Phi_\ell(j_{k-1}, j(E_1)) = 0 \\ \Phi_\ell(j_i, j_{i+1}) = 0 \quad \forall i \in \{1, \dots, k-2\} \end{array} \right\}$$

When generating isogeny path instances, we want the length k to be small enough to be efficient, but large enough to prevent meet-in-the-middle and collision search claw-finding type attacks [2, 39, 60], whose classical and quantum heuristic run times are $O(\ell^{k/2})$ and $O(\ell^{k/3})$ respectively. One might therefore take $k \approx 2\lambda$ as reasonable security trade-off. We note that the relation $\mathcal{R}_{\ell^k\text{-MODPOLY}}$ does not impose any restrictions on the underlying finite field. Hence, our method can apply to the CSIDH setting where j_i are defined over the prime field as long as the proof system supports the form of the prime. Our method can complement the efficient proof systems [15, 28] which have no restrictions on the degree of the witness.

Remark 2. Note that in this case, the isogeny may not necessarily be cyclic. In fact, the isogeny walk taken could indeed contain backtracking. In the applications we discuss in this work, this is not a problem, since an honest prover would honestly generate a non-backtracking isogeny of degree k , which would hence be cyclic. If one wishes to guarantee non-backtracking walks, this problem can be resolved by adding the requirement that $j_{i-1} \neq j_{i+1}$ for all i in $\{1, \dots, k-1\}$. We explain how to prove this with a cheap overhead in Appendix A.

3.2 High-Level Overview

The reader might wonder, what in particular does our isogeny representation achieve? What makes this relation so amenable to generic proof systems is its low-depth, highly *regular* decision circuit. That is, an arithmetic circuit C where $C(x, w) = 1$ if and only if $(x, w) \in \mathcal{R}_{\ell^k\text{-MODPOLY}}$. In this case, C may simply be a sequence of parallel evaluations of the modular polynomial on each pair of adjacent j -invariants. This allows us encode the relation in a highly compact (but equivalent) intermediate representation, to be fed into the proof system.

The general roadmap to utilising the generic proof systems is as follows:

1. Encode the relation $\mathcal{R}_{\ell^k\text{-MODPOLY}}$ and pair (x, w) into an equivalent RICS, denoted by $\mathcal{R}'_{\ell^k\text{-MODPOLY}}$ and (x', w') respectively.

2. Use a generic zkSNARK for R1CS (resp. arithmetic circuits) to argue the knowledge of a witness w' such that $(x', w') \in \mathcal{R}'_{\ell^k\text{-MODPOLY}}$.
3. The prover's knowledge of w' will imply the knowledge of w such that $(x, w) \in \mathcal{R}_{\ell^k\text{-MODPOLY}}$.

Since we have to perform field arithmetic over a quadratic extension field we can either work over the base field (where each \mathbb{F}_{p^2} -multiplication will dictate a series of underlying \mathbb{F}_p multiplications), or adapt the proof system implementation to be suitable for quadratic extensions. The security of the proof systems in question are independent of field choice, but the efficiency of Reed-Solomon based protocols is subject to a requirement. Namely, being capable of performing efficient FFT and IFFT operations. Broadly speaking, working over a field K , we require that K^\times contains a subgroup of order 2^m for an integer m such that $c \leq 2^m$, where $c = \max\{m, n\}$ for n variables and m constraints in a given R1CS. When working with isogenies, we typically choose primes of the form $p_1 = 2^a 3^b f - 1$, or $p_2 = 2^a 3^b f + 1$. It is clear that \mathbb{F}_{p_2} would satisfy the condition above, provided $m \leq a$, but \mathbb{F}_{p_1} would not, since $|\mathbb{F}_{p_1}^\times| = p_1 - 1 = 2(2^a 3^b f - 1)$. The first solution is to simply instantiate the proof system only over the base field with p_2 primes, however this admits several problems. Firstly, $\mathbb{F}_{p_1^2}$ operations are slightly more efficient. Since -1 is a non quadratic residue, $\mathbb{F}_{p_1^2} \cong \mathbb{F}(i)$ which allows for more efficient multiplication, inversion and squarings. Secondly, we want our protocol to be compatible with common choices of parameters, which typically use p_1 primes for efficiency reasons. Thus, we instantiate the proof system over the extension field, whose multiplicative order is $p^2 - 1 = (p - 1)(p + 1)$. This satisfies either choice of prime.

3.3 From Isogeny Relation to R1CS Instance

In order to apply our proof systems, we transform the modular polynomial relation into an R1CS with n variables and m constraints. Concretely, we consider an R1CS consisting of the statement $A, B, C \in \mathbb{F}_{p^2}^{m \times (n+1)}$ and a witness $z \in \mathbb{F}_{p^2}^{n+1}$ such that

$$Az \circ Bz = Cz.$$

In this formulation, A, B, C are public matrices which correspond to an instantiation of the language dependent on p, ℓ, k . The vector z consists of 1, the auxiliary input: j -invariants of the starting and ending curve, and the secret input: the j -invariant sequence (as well as intermediate variables dependent on the inputs). Each row of A, B, C will encode a linear constraint on the variables. One of these rows must encode the modular polynomial of level ℓ , $\Phi_\ell(j_i, j_{i+1}) = 0$, which shows that two adjacent j -invariants are isogenous. For representation compactness, we arrange the modular polynomial in the following form:

$$\begin{aligned}
 -1488XY(X + Y - 1488^{-1}XY) &= X^3 + Y^3 - 162000(X^2 + Y^2) + \\
 &8748000000(X + Y) + 40773375XY - 15746400000000 \quad (3)
 \end{aligned}$$

3.4 Optimization for RICS over \mathbb{F}_{p^2}

We then encode matrices A, B, C such that a row evaluates the equation above and performs intermediate variable consistency checks. Note that we can do far better than the naive approach, where each row of the matrices correspond to a single multiplication or addition of variables in z , and the entries of z contain every intermediate variable obtained. In loose terms, in RICS, each row can encode: *linear expression* \times *linear expression* = *linear expression*.

Suppose the isogeny path in question is of length k . If $k = 1, \ell = 2$ then by Eq. (2), we obtain:

$$z = (1 \ j_0 \ j_1 \ j_0^2 \ j_1^2 \ j_0^3 \ j_1^3 \ j_0 j_1)^T$$

with the matrices:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & c_4 & c_4 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c_0 & c_1 & c_1 & c_2 & c_2 & 1 & 1 & c_3 \end{bmatrix}$$

where

$$\begin{aligned} c_0 &= -157464000000000 & c_1 &= 8748000000 & c_2 &= -162000 \\ c_3 &= 40773375 & c_4 &= 1488, \end{aligned}$$

where the c_i 's are derived from Eq. (3). The first 5 rows provide consistency checks on each variable, including square, cube, and multiplication. The last row checks the evaluation of the polynomial Eq. (3). Now we can extend this to a path of length $k > 1$, for each j -invariant j_i , we will introduce an additional 4 variables (including input): $j_i, j_i^2, j_i^3, j_{i-1}j_i$. We note that the squarings and cubings for each j -invariant need only be checked once. Hence, we obtain $n := 4k + 3$ variables.

For each j -invariant in the sequence (including j_0) there will be 2 constraints for squaring and cubing consistency checks. For each adjacent pair j_{i-1}, j_i , there will be 2 constraints: one checking consistency of the variable $j_{i-1}j_i$, and one the evaluation of the modular polynomial. This gives us $m := 4k + 2$ constraints.

3.5 Optimization for Lifting to $\mathbb{F}_p \times \mathbb{F}_p$

This subsection presents several techniques to reduce the overhead to lift arithmetic over a quadratic field to a vector space of the prime field. We consider a quadratic field $\mathbb{F}_{p^2} \cong \mathbb{F}_p[\alpha]$ where $\alpha^2 = d$ for some non-square $d \in \mathbb{F}_p$.

The motivation is that, generally, the j -invariant of an elliptic curve is taken over \mathbb{F}_{p^2} while many proof systems natively support arithmetic over a prime field. The impact to performance of either choice was unclear. Indeed, arithmetic computations over $\mathbb{F}_p[\alpha]$ can be viewed as arithmetic computations over an \mathbb{F}_p -vector space natively. That is, for $x_1, x_2, y_1, y_2 \in \mathbb{F}_p$ to represent $x_1 + x_2\alpha \in \mathbb{F}_p[\alpha]$, by mapping $x_1 + x_2\alpha$ to (x_1, x_2) the addition is $(x_1 + y_1, x_2 + y_2)$ and the multiplication is $(x_1y_1 + x_2y_2d, x_1y_2 + x_2y_1)$. Naively, this results in 4 (variable) \mathbb{F}_p -multiplications for one (variable) \mathbb{F}_{p^2} -multiplication (i.e. $x_1x_2, y_1y_2, x_1y_2, x_2y_1$). In fact, with a few well-known tricks, this can be done more efficiently:

Arithmetic. We start with multiplications.

- $u_1 = x_1y_1$
- $u_2 = y_2y_2$
- $u_3 = (x_1 + x_2)(y_1 + y_2)$, then
- $x_1y_1 + x_2y_2d = u_1 + u_2d$
- $(x_1y_2 + x_2y_1) = u_3 - u_1 - u_2$

Now, there are only 3 (variable) \mathbb{F}_p -multiplications. The benefit to this depends on the proof system to be used. In many proof systems, it is much more expensive to verify a (variable) multiplication relation than a (variable) linear relation.

Let $x + y\alpha \in \mathbb{F}_p[\alpha]$, there is a trick for variable squaring:

- $u_1 = xy$
- $u_2 = (x + y)(x + yd)$, then
- $(x^2 + y^2i^2) = u_2 - (d + 1)u_1$
- $2xy = 2u_1$

Application to R1CS Matrices. Now we can apply the abovementioned techniques to our R1CS matrices. Recall that in Sect. 3.4, we have a witness vector z over $\mathbb{F}_p \times \mathbb{F}_p^7$. To lift it into \mathbb{F}_p , we firstly naturally embed it into $\mathbb{F}_p \times \mathbb{F}_p^{14}$. We explain how to build a submatrices and introduce intermediate variables for each constraint as follows. As an abuse of notation, given an element $x := a + b\alpha \in \mathbb{F}_p[\alpha]$, we refer to a as $\text{Re}(x)$ and b as $\text{Im}(x)$ respectively.

Squaring. For the squaring relation, it is fairly simple. Take the subvector $(1, \text{Re}(x), \text{Im}(x), \text{Re}(x^2), \text{Im}(x^2))$ for instance, the corresponding submatrices for this constraint are respectively

$$\begin{bmatrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & d & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2^{-1}(d + 1) \end{bmatrix},$$

which represents $2\text{Re}(x)\text{Im}(x) = \text{Im}(x^2)$ and $(\text{Re}(x) + \text{Im}(x))(\text{Re}(x) + d\text{Im}(x)) = \text{Re}(x^2) + 2^{-1}(d + 1)\text{Im}(x^2)$, resp.

Multiplication. For the multiplication relation, we need an additional variable u over \mathbb{F}_p . We take the subvector $(1, \text{Re}(x), \text{Im}(x), \text{Re}(y), \text{Im}(y), u, \text{Re}(xy), \text{Im}(xy))$ for instance. The corresponding submatrices for this constraint are respectively

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -d & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -d & 1 \end{bmatrix},$$

which represents $\text{Im}(x)\text{Im}(y) = u$, $\text{Re}(x)\text{Re}(y) = \text{Re}(xy) - ud$, and $(\text{Re}(x) + \text{Im}(x))(\text{Re}(y) + \text{Im}(y)) = \text{Im}(xy) + \text{Re}(x)\text{Re}(y) + u$, respectively.

- and isogeny-based protocols with primes of the form $p = 2^a f \pm 1$ operating over \mathbb{F}_{p^2} .

Once an isogeny path has been obtained, it is straightforward to obtain either R1CS instance given the methods described in Sect. 3.4 and Sect. 3.5. We leave the manner in which the isogeny paths are computed open to a more detailed implementation. One such approach would be to use optimized SIDH implementations [5, 26], with some modifications needed to support p_2 primes. Note that since $p_2 \equiv 1 \pmod{4}$, the curves of j -invariant 0,1728 are not supersingular. In this setting, one can find a starting curve by using a root of the Hilbert class polynomial mod p [20, Sec 3.2]. The public parameters p, ℓ, k are sufficient for a verifier to efficiently construct the R1CS matrices A, B, C offline, which minimises the communication and storage cost.

In evaluating performance for comparison with [6], we have included the standard SIKE parameters, but also include primes of comparable parameters of the p_2 form in order to compare performance over different base fields, which should offer equivalent security at the cost of slightly reduced performance of isogeny path computation. These primes are the smallest primes $p_2 = 2^a 3^b f + 1$ such that for a corresponding SIKE prime $p_1 = 2^{a'} 3^{b'} f' - 1$ we have that $a \geq a'$, $b \geq b'$ and $f' \geq f$. Due to the flexibility of the underlying proof systems, the protocol can operate over arbitrary choices of k , and primes of this form. We have fixed the path length k to the corresponding lengths of the Secuer PoK 2-isogeny path, which is of sufficient length to guarantee a uniformly distributed end point, assuming a random walk. In fact, this is a conservative choice. It is conjectured that non-backtracking walks can converge to the stationary distribution in shorter walks than compared to [6]. See [58, Conjecture 4.3]. We stress that in many applications, uniform mixing is not necessary. In order to guarantee minimal security, the path's length must be approximately 2λ . For the parameters and results of our proof for minimally secure path lengths, see Sect. 4.2.

Remark 3. The choice of benchmarking this protocol with parameters obtained from the now defunct SIKE may seem somewhat arbitrary. We do so to compare our results to [6], whose implementation is limited to SIKE primes. A pragmatic course of action might be to determine concrete parameters that are practical and secure in the setting of isogeny commitments and hashing.

4 Implementation and Evaluation

In evaluating the performance of our isogeny proof of knowledge, we considered protocols which support finite fields of arbitrary prime characteristic⁸, that are statistical zero-knowledge, plausibly post-quantum and transparent (see Table 2).

Virgo and Orion [62, 64] do satisfy these properties. However, we excluded them from our testing as their implementation does not easily support generic

⁸ Subject to FFT performance conditions.

Table 1. Our parameter sets for the evaluation of isogeny PoK in R1CS representation.

p	k	Variant	R1CS Param.		Security Level	
			n	m		
p434	$2^{216}3^{137} - 1$	705	\mathbb{F}_{p^2}	2823	2822	$\lambda = 128$
p441+	$2^{218}3^{138}37 + 1$		\mathbb{F}_p	7759	7758	
p503	$2^{250}3^{159} - 1$	774	\mathbb{F}_{p^2}	3099	3098	$\lambda = 128$
p509+	$2^{252}3^{159}31 + 1$		\mathbb{F}_p	8518	8517	
p610	$2^{305}3^{192} - 1$	1010	\mathbb{F}_{p^2}	4043	4044	$\lambda = 192$
p619+	$2^{307}3^{192}119 + 1$		\mathbb{F}_p	11114	11113	
p751	$2^{372}3^{239} - 1$	1280	\mathbb{F}_{p^2}	5123	5122	$\lambda = 256$
p761+	$2^{372}3^{239}701 + 1$		\mathbb{F}_p	14084	14083	

fields, but we hope to include them in future testing. Theoretically, Virgo performs well for low-depth, uniform circuits such as our own.

The state-of-the-art is given by Ligerio++; a protocol that combines aspects of Virgo and Ligerio, trading-off marginally higher verification times for faster prover times than Aurora, with comparable proof sizes. However, it does not have any open source implementations. Brakedown, Shockwave [44]; and the recent LaBRADOR [16] are candidates of interest. However, they do not yet offer zero-knowledge. There are no clear obstructions to them achieving zero-knowledge, and provide promising results, so are worth considering in future lines of work.

Table 2. Asymptotic cost various transparent, post-quantum, zero-knowledge generic proof systems, applied to an arithmetic circuit of N gates, n inputs, and depth D over a fixed field.

	Prover time	Verifier time	Proof size
Limbo [33]	$O(N)$	$O(N)$	$O(N)$
Ligerio [3]	$O(N \log N)$	$O(N)$	$O(\sqrt{N})$
Aurora [10]	$O(N \log N)$	$O(N)$	$O(\log^2 N)$
Virgo [64]	$O(N + n \log n)$	$O(D \log N + \log^2 n)$	$O(D \log N + \log^2 n)$
Ligerio++ [17]	$O(N \log N)$	$O(N)$	$O(\log^2 N)$
Orion [62]	$O(N)$	$O(\log^2 N)$	$O(\log^2 N)$

Implementation. As a proof of concept, we evaluate the performance of our isogeny proof of knowledge via:

- **Aurora** and **Ligero** through a fork of `libiop`⁹, modified to support larger prime fields and quadratic field extensions. Ligero’s original implementation is closed source, but an adaptation is included in `libiop`. While originally designed for arithmetic circuit satisfiability, `libiop`’s implementation supports R1CS instead, at claimed *no extra cost*.
- **Limbo**, through an implementation obtained via private correspondence (the publicly available implementation is only available for binary fields). Limbo is interfaced with our R1CS instances directly, with an arithmetic circuit that evaluates $Az \circ Bz - Cz$ and then checking that the resulting vector equals to zero.

Aurora and Ligero are directly tested with R1CS instances of size given in Table 1. We separate the results for the standard SIKE parameters for direct comparison with SECUER PoK, and include a second table of results (Table 4) for the smooth primes which operate over \mathbb{F}_p . Limbo is directly interfaced to prove the given R1CS instance in a manner described in Sect. 2.5.

4.1 Comparison to SECUER PoK

To make a comparison, we include the results from the **SECUER PoK** [6], through the reference implementation¹⁰. The SECUER PoK is a direct proof of knowledge for a relaxed notion of the relation $\mathcal{R}_{2^k\text{-ISO}\text{PATH}}$, so provides comparison as a tailored protocol to our results from applying generic proof systems.

Remark 4. In the previous version, our implementations are inconsistent with [6] regarding the walk length. The SECUER PoK reports results for walks of sufficient length in order to guarantee uniform mixing in the supersingular isogeny graph. For consistency, we now evaluate our results based on the same parameters. This is a desired feature in the setup ceremony protocol introduced in their work. However, this is not a strict requirement in isogeny-based protocols. We include an additional set of results in Sect. 4.2 which include results for walk lengths which are minimally secure for the respective security levels, which may be of interest in wider applications.

Results. The experiments are run on a Intel® Core™ i9-9900 CPU @ 3.10GHz. The benchmarks include only single-threaded results as the `libiop` package does not properly implement multi-threading and did not provide accurate results. Nevertheless, Aurora and Ligero should reflect similar optimizations to that of [6] from a well supported multi-threaded implementation, as the protocols are well suited to parallelisation. In particular, the protocols run parallel compositions of the proof in order to achieve necessary soundness level (Table 3).

⁹ Original source code available at <https://github.com/scipr-lab/libiop>. Our fork can be found at <https://github.com/levanin/libiop-fp2>.

¹⁰ Source code available at <https://github.com/trusted-isogenies/SECUER-pok>.

Table 3. Table of results comparing several generic proof systems operating over \mathbb{F}_{p^2} for the R1CS instantiation of $\mathcal{R}_{2^k\text{-MP}}$, and the isogeny SECEUR PoK in [6]. Security level and walk length is set according to Table 1 and P, V, S correspond to proof time, verification time, and proof size respectively. Results displayed are for single-threaded performance.

Parameter	Our Work			SECEUR PoK	
	Aurora	Ligero	Limbo		
p434	P	4,204 ms	1,479 ms	1,073 ms	12,369 ms
	V	378 ms	1,899 ms	874 ms	1,399 ms
	S	277 kB	3,281 kB	8,133 kB	191 kB
p503	P	4,944 ms	1,722 ms	1,379 ms	19,296 ms
	V	440 ms	2,171 ms	1,146 ms	2,173 ms
	S	313 kB	3,778 kB	10,335 kB	216 kB
p610	P	6,457 ms	3,331 ms	3,156 ms	60,915 ms
	V	888 ms	3,102 ms	2,616 ms	6,646 ms
	S	570 kB	4,568 kB	24,427 kB	404 kB
p751	P	15,070 ms	5,243 ms	7,702 ms	141,043 ms
	V	2,383 ms	13,509 ms	6,587 ms	15,931 ms
	S	852 kB	11,302 kB	50,670 kB	663 kB

Table 4. Table of results comparing generic proof systems operating over \mathbb{F}_p for the projected R1CS instantiation of $R_{2^k\text{-MP}}$ operating over fields with characteristic of the form $2^a 3^b f + 1$. Security level and walk lengths set according to Table 1. Results displayed are for single threaded performance.

Parameter	Aurora	Ligero	Limbo	
p441+	P	2,313 ms	879 ms	1,037 ms
	V	158 ms	1017 ms	835 ms
	S	152 kB	2,803 kB	11,217 kB
p509+	P	5,999 ms	1301 ms	1,304 ms
	V	455 ms	1370 ms	1,066 ms
	S	214 kB	3,402 kB	14,205 kB
p619+	P	9,424 ms	2,822 ms	2,962 ms
	V	895 ms	2,030 ms	2,451 ms
	S	409 kB	4,149 kB	33,746 kB
p761+	P	12,555 ms	2,873 ms	7,062 ms
	V	1,651 ms	4,464 ms	5,823 ms
	S	687 kB	7,212 kB	70,018 kB

We see that Aurora, the best overall performer, provides a 3-12 times improvement to proof and verification times compared to SECUER PoK, with 0-30% increase in proof length. If we consider smooth primes which allow for operation over \mathbb{F}_p , Aurora allows for similar improvements to proof and verification times but with smaller proofs than SECUER PoK when compared with parameters of similar bit length. Limbo, as expected, performs well for smaller parameters at the cost of much longer proof lengths. Conversely, Ligerio is better suited to larger parameters than Limbo but still suffers from long proofs. These results should serve as evidence to support the choice of Aurora as a platform for this application.

4.2 Identification Scheme for Moderate Length Walks

Our proof system may also serve as an identification scheme to validate a public key (E, E') , where the prover can use our zkSNARK construction to demonstrate their knowledge of a walk from E to E' , of sufficient length to resist the most efficient generic algorithm for recovering the secret isogeny (i.e. the claw finding algorithm). In this section, we demonstrate the effectiveness of our proof system in this regard.

We show in Table 5 the R1CS parameter set (m, n) over \mathbb{F}_p and \mathbb{F}_{p^2} and the isogeny walk length k with respect to the security parameter λ . A concrete result is given in Tables 6 and 7 regarding the prover time, the verifier time and the proof size for different forms of the primes.

Table 5. Our R1CS parameter set (m, n) over \mathbb{F}_p and \mathbb{F}_{p^2} and the isogeny walk length k with respect to the security parameter λ and the prime p .

	p	k	Variant	R1CS Param.		Security Level
				n	m	
p434	$2^{216}3^{137} - 1$	216	\mathbb{F}_{p^2}	867	866	$\lambda = 128$
p441+	$2^{218}3^{138}37 + 1$		\mathbb{F}_p	2380	2379	
p503	$2^{250}3^{159} - 1$	250	\mathbb{F}_{p^2}	1003	1002	$\lambda = 128$
p509+	$2^{252}3^{159}31 + 1$		\mathbb{F}_p	2754	2753	
p610	$2^{305}3^{192} - 1$	305	\mathbb{F}_{p^2}	1223	1222	$\lambda = 192$
p619+	$2^{307}3^{192}119 + 1$		\mathbb{F}_p	3359	3358	
p751	$2^{372}3^{239} - 1$	372	\mathbb{F}_{p^2}	1491	1490	$\lambda = 256$
p761+	$2^{372}3^{239}701 + 1$		\mathbb{F}_p	4096	4095	

Table 6. Table of results comparing several generic proof systems operating over \mathbb{F}_{p^2} for the R1CS instantiation of $\mathcal{R}_{2^k\text{-MP}}$ without relaxations. The soundness/zero-knowledge security level is set according to Table 5 and P, V, S correspond to proof time, verification time, and proof size respectively. Results displayed are for single-threaded performance.

Parameter	Our Work		
	Aurora	Ligero	Limbo
p434	P 934 ms	587 ms	354 ms
	V 99 ms	847 ms	273 ms
	S 194 kB	1,849 kB	2,598 kB
p503	P 1,138 ms	686 ms	479 ms
	V 114 ms	959 ms	380 ms
	S 219 kB	2,127 kB	3,456 kB
p610	P 3,175 ms	2,488 ms	989 ms
	V 472 ms	2614 ms	818 ms
	S 517 kB	4,084 kB	7,607 kB
p751	P 3,882 ms	1,951 ms	2,131 ms
	V 824 ms	6407 ms	1,793 ms
	S 828 kB	6,394 kB	15,104 kB

Table 7. Table of results comparing generic proof systems operating over \mathbb{F}_p for the projected R1CS instantiation of $R_{2^k\text{-MP}}$ operating over fields with characteristic of the form $2^a 3^b f + 1$. Soundness/zero-knowledge security levels set according to Table 5. Results displayed are for single threaded performance.

Parameter	Aurora	Ligero	Limbo
p441+	P 1,216 ms	427 ms	330 ms
	V 98 ms	493 ms	264 ms
	S 166 kB	1,733 kB	3,496 kB
p509+	P 1,440 ms	537 ms	438 ms
	V 120 ms	603 ms	342 ms
	S 182 kB	1,967 kB	4,657 kB
p619+	P 2,287 ms	1,130 ms	922 ms
	V 239 ms	849 ms	746 ms
	S 338 kB	2,414 kB	10,327 kB
p761+	P 3,030 ms	1,044 ms	1,938 ms
	V 431 ms	1,951 ms	1,594 ms
	S 551 kB	4,004 kB	20,588 kB

5 Conclusion

In conclusion, we show that generic proof systems are competitive when applied to isogeny-based relations, by giving a proof of concept for an isogeny proof of knowledge using a compact RICS instance, whose security is based on the underlying proof systems. Our best experimental result shows an order of magnitude improvement for prover and verifier time compared to the state-of-the-art tailor-made isogeny protocol, SECUER PoK.

A Remark on Signatures. Several post-quantum signature schemes have been proposed by applying MPCitH proof systems to PRFs, such as [7, 12, 32, 63]. The approach follows one of two processes, given a uniform secret key k :

1. The public key is y such that $f(k) = y$ for a one-way function f . A signature corresponds to a non-interactive proof that “I know a k such that $f(k) = y$ ” where the message m is incorporated into the randomness of the challenges.
2. The public key is $\text{PRF}_k(0^\lambda)$, and a signature is then an evaluation of $\text{PRF}_k(m)$ attached with a proof that “I know a k such that I can compute both $\text{PRF}_k(m)$ and $\text{PRF}_k(0^\lambda)$ ”.

Given a secure PRF, the latter approach is somewhat agnostic to the proof system in question. However, in the former case, it is unclear that proofs obtained from the BCS transform applied to IOPs can yield a secure signature scheme analogous to Fiat-Shamir applied to Σ -protocols. Some works [35, 40] indicate the non-malleability or *simulation extractability* is an important notion in the security of this construction. Simulation extractability provides that a malicious prover cannot forge a valid proof without knowledge of the witness, even after seeing polynomially many valid proofs. In particular, this notion seems to yield a direct reduction to EUF-CMA. To this date, the security of the BCS transform with messages incorporated into the verifier’s randomness lacks sufficient analysis, and it is unclear as to what property is necessary and sufficient in order to construct signatures by (1). If this is achieved, it is straightforward to convert the isogeny proof of knowledge into a signature scheme based on the hardness isogeny path-finding, where the one-way function is essentially the CGL hash function.

Acknowledgements. This work is a result of a collaboration at the University of Auckland, and is supported by the Ministry for Business, Innovation and Employment of New Zealand; the Cyber Security Research Flanders with reference number VR20192203; the Defense Advanced Research Projects Agency (DARPA); and the Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. FA8750-19-C-0502.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any of the funders. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

We would like to acknowledge and thank Steven Galbraith for sharing his valuable supervision and insight on the project; as well as Cyprien Delpuch de Saint Guilhem and Titouan Tanguy for their support with the Limbo implementation. We thank Luca de Feo for sharing his valuable suggestions and opinions to improve the clarity and correctness of this work.

A Preventing Backtracking

In some applications, one might want a guarantee that the isogeny proven is cyclic, which in our setting, is equivalent to showing that the isogeny walk is non-backtracking. That is, a walk which does not immediately traverse the same edge twice.

Theorem 2. *An isogeny $\phi : E_0 \rightarrow E_k$ of degree 2^k is cyclic if and only if ϕ 's decomposition into 2-isogenies as a walk on the supersingular isogeny graph is non-backtracking.*

Proof. See [24, Prop. 1][34, Prop. 3.2]

In the modular polynomial relation we introduce, we do not provide any guarantee that our isogeny is non-backtracking (and hence cyclic). However, with minor overhead, it is possible to add this requirement. Observe that, given an isogeny walk from E_0 to E_k of length k , with a j -invariant sequence j_0, \dots, j_k , a backtracking walk implies that there exists an $i \in \{1, \dots, k-1\}$ such that $j_{i-1} = j_{i+1}$. So it suffices to show that

$$\delta_i = j_{i-1} - j_{i+1} \neq 0 \text{ for all } i \in \{1, \dots, k-1\}.$$

One can realise inequality in an arithmetic circuit with the following process: given two numbers a, b , we may show that they are distinct if and only if there exists an inverse of $(a - b)$ over the field. Alternatively, there exists c such that $(a - b) \cdot c = 1$. The inverse $c := (a - b)^{-1}$ can be precomputed by the prover and given as a part of the input.

We can perform an additional optimization step to minimise the number of precomputed inverses for the prover, the calculation of which is expensive. Indeed, the prover can accumulate the product of the difference terms δ_i , and check that the product is nonzero. In particular, our resulting conditions to prevent backtracking are that:

1. Compute $\delta = \prod \delta_i = \prod_{i=1}^{k-1} (j_{i-1} - j_{i+1})$.
2. Input δ' such that $\delta\delta' = 1$,

where the δ term will be non-zero if and only if all δ_i are non-zero, which is true if (but not only if) the walk is non-backtracking. We note that this check will also prevent the use of 2-cycles (with two distinct edges), which may be cyclic, but are of little consequence in practice.

It is straightforward to add these constraints to our previous RICS instance. In the \mathbb{F}_{p^2} setting, this would add an additional $k-1$ constraints and variables for

the product check; and one constraint and variable (the inverse given as input) for the inverse check. This version yields $5k + 3$ variables and $5k + 2$ constraints in total, which means only a 25% overhead if compared to the original instance size. We expect this to subject only a minor performance cost, as the protocol scales well with instance size (as seen in the difference between Sect. 4.2 and Sect. 4.1).

References

1. Abdalla, M., Eisenhofer, T., Kiltz, E., Kunzweiler, S., Riepel, D.: Password-authenticated key exchange from group actions. *Cryptology ePrint Archive*, Paper 2022/770 (2022). <https://eprint.iacr.org/2022/770>, <https://eprint.iacr.org/2022/770>
2. Adj, G., Cervantes-Vázquez, D., Chi-Domínguez, J.J., Menezes, A., Rodríguez-Henríquez, F.: On the cost of computing isogenies between supersingular elliptic curves. In: Cid, C., Jacobson Jr: M.J. (eds.) SAC 2018. LNCS, vol. 11349, pp. 322–343. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-10970-7_15
3. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Liger: lightweight sub-linear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 2087–2104. ACM Press (Oct/Nov 2017)
4. Atapoor, S., Bagheri, K., Cozzo, D., Pedersen, R.: Csi-shark: csi-fish with sharing-friendly keys. *Cryptology ePrint Archive*, Paper 2022/1189 (2022). <https://eprint.iacr.org/2022/1189>
5. Azarderakhsh, R., et al.: Supersingular isogeny key encapsulation (SIKE). Submission to the NIST Post-Quantum Standardization Project (2017). <https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/round-4/submissions/SIKE-spec.pdf>
6. Basso, A., et al.: Supersingular curves you can trust. *Cryptology ePrint Archive*, Paper 2022/1469 (2022). <https://eprint.iacr.org/2022/1469>
7. Baum, C., de Saint Guilhem, C.D., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: short and fast signatures from AES. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12710, pp. 266–297. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75245-3_11
8. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) ICALP 2018. LIPIcs, vol. 107, pp. 14:1–14:17. Schloss Dagstuhl (Jul 2018)
9. Ben-Sasson, E., et al.: Zerocash: decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 459–474. IEEE Computer Society Press (May 2014)
10. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_4
11. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_2

12. Beullens, W., Delpech de Saint Guilhem, C.: LegRoast: efficient post-quantum signatures from the legendre PRF. In: Ding, J., Tillich, J.-P. (eds.) PQCrypto 2020. LNCS, vol. 12100, pp. 130–150. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_8
13. Beullens, W., Dobson, S., Katsumata, S., Lai, Y.F., Pintore, F.: Group signatures and more from isogenies and lattices: Generic, simple, and efficient. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 95–126. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-07085-3_4
14. Beullens, W., Katsumata, S., Pintore, F.: Calamari and Falaf: Logarithmic (Linkable) ring signatures from isogenies and lattices. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 464–492. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_16
15. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 227–247. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_9
16. Beullens, W., Seiler, G.: Labrador: Compact proofs for r1cs from module-sis. Cryptology ePrint Archive, Paper 2022/1341 (2022). <https://eprint.iacr.org/2022/1341>
17. Bhadauria, R., Fang, Z., Hazay, C., Venkatasubramanian, M., Xie, T., Zhang, Y.: Liger++: a new optimized sublinear IOP. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020, pp. 2025–2038. ACM Press (Nov 2020)
18. Booher, J., et al.: Failing to hash into supersingular isogeny graphs. Cryptology ePrint Archive, Report 2022/518 (2022). <https://eprint.iacr.org/2022/518>
19. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_12
20. Bröker, R.: Constructing elliptic curves of prescribed order (2008)
21. Burdges, J., De Feo, L.: Delay encryption. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 302–326. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_11
22. Castryck, W., Decru, T.: An efficient key recovery attack on sidh (preliminary version). Cryptology ePrint Archive, Paper 2022/975 (2022). <https://eprint.iacr.org/2022/975>
23. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 395–427. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_15
24. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. *J. Cryptol.* **22**(1), 93–113 (2009)
25. Chavez-Saab, J., Rodríguez-Henríquez, F., Tibouchi, M.: Verifiable isogeny walks: towards an isogeny-based postquantum VDF. In: AlTawy, R., Hülsing, A. (eds.) SAC 2021. LNCS, vol. 13203, pp. 441–460. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99277-4_21
26. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 572–601. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_21
27. De Feo, L., Dobson, S., Galbraith, S.D., Zobernig, L.: SIDH proof of knowledge. Cryptology ePrint Archive, Report 2021/1023 (2021). <https://eprint.iacr.org/2021/1023>



28. De Feo, L., Galbraith, S.D.: SeaSign: compact isogeny signatures from class group actions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 759–789. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_26
29. De Feo, L., Jao, D., Plüt, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Cryptology ePrint Archive, Report 2011/506 (2011). <https://eprint.iacr.org/2011/506>
30. De Feo, L., Jao, D., Plüt, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptology* **8**(3), 209–247 (2014)
31. De Feo, L., Kohel, D., Leroux, A., Petit, C., Wesolowski, B.: SQISign: compact post-quantum signatures from quaternions and isogenies. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12491, pp. 64–93. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_3
32. de Saint Guilhem, C.D., De Meyer, L., Orsini, E., Smart, N.P.: BBQ: Using AES in Picnic Signatures. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 669–692. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38471-5_27
33. de Saint Guilhem, C., Orsini, E., Tanguy, T.: Limbo: Efficient zero-knowledge MPCitH-based arguments. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021, pp. 3022–3036. ACM Press (Nov 2021)
34. Doliskani, J., Pereira, G.C.C.F., Barreto, P.S.L.M.: Faster cryptographic hash function from supersingular isogeny graphs. Cryptology ePrint Archive, Paper 2017/1202 (2017). <https://eprint.iacr.org/2017/1202>, <https://eprint.iacr.org/2017/1202>
35. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat-Shamir transform. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34931-7_5
36. Feneuil, T., Joux, A., Rivain, M.: Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. Cryptology ePrint Archive, Report 2022/188 (2022). <https://eprint.iacr.org/2022/188>
37. Feneuil, T., Maire, J., Rivain, M., Vergnaud, D.: Zero-knowledge protocols for the subset sum problem from MPC-in-the-head with rejection. Cryptology ePrint Archive, Report 2022/223 (2022). <https://eprint.iacr.org/2022/223>
38. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
39. Galbraith, S.D.: Constructing isogenies between elliptic curves over finite fields. *LMS J. Comput. Math.* **2**, 118–138 (1999)
40. Ganesh, C., Khoshakhlagh, H., Kohlweiss, M., Nitulescu, A., Zając, M.: What Makes Fiat-Shamir zkSNARKs (Updatable SRS) Simulation Extractable? In: Galdi, C., Jarecki, S. (eds.) SCN 2022. LNCS, pp. 735–760. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-14791-3_32
41. Gennaro, R., Minelli, M., Nitulescu, A., Orrù, M.: Lattice-based zk-SNARKs from square span programs. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018, pp. 556–573. ACM Press (Oct 2018)
42. Ghantous, W., Katsumata, S., Pintore, F., Veroni, M.: Collisions in supersingular isogeny graphs and the sidh-based identification protocol. Cryptology ePrint Archive, Paper 2021/1051 (2021). <https://eprint.iacr.org/2021/1051>
43. Ghantous, W., Pintore, F., Veroni, M.: Collisions in supersingular isogeny graphs and the SIDH-based identification protocol. Cryptology ePrint Archive, Report 2021/1051 (2021). <https://eprint.iacr.org/2021/1051>

44. Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and post-quantum SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/1043 (2021). <https://eprint.iacr.org/2021/1043>
45. Ishai, Y.: Zero-knowledge proofs from information-theoretic proof systems. Zkproofs Blog (2020)
46. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC, pp. 21–30. ACM Press (Jun 2007)
47. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.* **39**(3), 1121–1152 (2009)
48. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
49. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018, pp. 525–537. ACM Press (Oct 2018)
50. Lai, Y.-F., Galbraith, S.D., Delpech de Saint Guilhem, C.: Compact, efficient and UC-secure isogeny-based oblivious transfer. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 213–241. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_8
51. Leroux, A.: A new isogeny representation and applications to cryptography. Cryptology ePrint Archive, Report 2021/1600 (2021). <https://eprint.iacr.org/2021/1600>
52. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A Direct Key Recovery Attack on SIDH. In: Eurocrypt 2023. Lyon, France (Apr 2023). <https://hal.science/hal-04023441>
53. Mula, M., Murru, N., Pintore, F.: Random sampling of supersingular elliptic curves. Cryptology ePrint Archive, Report 2022/528 (2022). <https://eprint.iacr.org/2022/528>
54. Onuki, H.: On oriented supersingular elliptic curves. *Finite Fields Appl.* **69**, 101777 (2021)
55. Pizer, A.K.: Ramanujan graphs and Hecke operators. *Bull. Am. Math. Soc.* **23**(1), 127–137 (1990). <https://www.ams.org/bull/1990-23-01/S0273-0979-1990-15918-X/>
56. Robert, D.: Breaking sidh in polynomial time. Cryptology ePrint Archive, Paper 2022/1038 (2022). <https://eprint.iacr.org/2022/1038>
57. Silverman, J.H.: *The Arithmetic of Elliptic Curves*, Graduate Texts in Mathematics, vol. 106. Springer, New York (2009). <http://link.springer.com/10.1007/978-0-387-09494-6>
58. Sterner, B.: Commitment schemes from supersingular elliptic curve isogeny graphs. *Math. Cryptology* **1**(2), 40–51 (Mar 2022). <https://journals.flvc.org/mathcryptology/article/view/130656>
59. Thaler, J.: Proofs, arguments, and zero-knowledge (2020)
60. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. *J. Cryptol.* **12**(1), 1–28 (1999)
61. Washington, L.C.: *Elliptic curves: number theory and cryptography*. Chapman and Hall/CRC (2008)
62. Xie, T., Zhang, Y., Song, D.: Orion: zero knowledge proof with linear prover time. Cryptology ePrint Archive, Paper 2022/1010 (2022). <https://eprint.iacr.org/2022/1010>

63. Zaverucha, G., et al.: Tech. rep., National Institute of Standards and Technology (2020). <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>
64. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. Cryptology ePrint Archive, Report 2019/1482 (2019). <https://eprint.iacr.org/2019/1482>



Low Memory Attacks on Small Key CSIDH

Jesús-Javier Chi-Domínguez¹ , Andre Esser¹ , Sabrina Kunzweiler^{2,3} ,
and Alexander May³ 

¹ Technology Innovation Institute, Abu Dhabi, UAE

{jesus.dominguez, andre.esser}@tii.ae

² Univ. Bordeaux, CNRS, Bordeaux INP, Inria, Talence, France

sabrina.kunzweiler@math.u-bordeaux.fr

³ Ruhr University Bochum, Bochum, Germany

alex.may@rub.de

Abstract. Despite recent breakthrough results in attacking SIDH, the CSIDH protocol remains a secure post-quantum key exchange protocol with appealing properties. However, for obtaining efficient CSIDH instantiations one has to resort to small secret keys. In this work, we provide novel methods to analyze small key CSIDH, thereby introducing the representation method—that has been successfully applied for attacking small secret keys in code- and lattice-based schemes—also to the isogeny-based world.

We use the recently introduced Restricted Effective Group Actions (REGA) to illustrate the analogy between CSIDH and Diffie-Hellman key exchange. This framework allows us to introduce a REGA-DLOG problem as a level of abstraction to computing isogenies between elliptic curves, analogous to the classic discrete logarithm problem. This in turn allows us to study REGA-DLOG with ternary key spaces such as $\{-1, 0, 1\}^n$, $\{0, 1, 2\}^n$ and $\{-2, 0, 2\}^n$, which lead to especially efficient, recently proposed CSIDH instantiations. The best classic attack on these key spaces is a Meet-in-the-Middle algorithm that runs in time $3^{0.5n}$, using also $3^{0.5n}$ memory.

We first show that REGA-DLOG with ternary key spaces $\{0, 1, 2\}^n$ or $\{-2, 0, 2\}^n$ can be reduced to the ternary key space $\{-1, 0, 1\}^n$.

We further provide a heuristic time-memory tradeoff for REGA-DLOG with keyspace $\{-1, 0, 1\}^n$ based on Parallel Collision Search with memory requirement M that under standard heuristics runs in time $3^{0.75n}/M^{0.5}$ for all $M \leq 3^{n/2}$. We then use the representation technique to heuristically improve to $3^{0.675n}/M^{0.5}$ for all $M \leq 3^{0.22n}$, and further provide more efficient time-memory tradeoffs for all $M \leq 3^{n/2}$.

Although we focus in this work on REGA-DLOG with ternary key spaces for showing its efficacy in providing attractive time-memory tradeoffs, we also show how to use our framework to analyze larger key spaces $\{-m, \dots, m\}^n$ with $m = 2, 3$.

Keywords: Isogeny · Time-Memory Trade-off · Representation Technique

1 Introduction

In the pre-quantum era the Diffie-Hellman protocol plays a paramount role in securely exchanging secret keys. Diffie-Hellman shows its outstanding performance when instantiated with sufficiently generic elliptic curve groups with prime order q , since for solving the discrete logarithm in these groups on classical computers only generic algorithms with square-root time complexity $\Theta(\sqrt{q})$ are known.

Such a complexity allows for extremely efficient instantiations that provide e.g. 128 bit classical security for 256-bit group order q . Since Shor’s algorithm [42] generically breaks discrete logarithms in every commutative group of order q in time polynomial in $\log q$, Diffie-Hellman unfortunately becomes completely insecure in a quantum world.

The current post-quantum substitutes for key exchange primarily stem from lattice problems, like Kyber [10], or from decoding problems, like McEliece [3, 34]. However, in both cases we have already classical algorithms that are below square root complexity [5, 39]. As a consequence, lattice- and code-based schemes can inherently not achieve the efficiency of the Diffie-Hellman protocol. For exploiting smallness of secret keys, the *representation technique* has been quite successfully applied first in the coding world [6, 11, 20, 31, 33], and then subsequently also for lattice-based schemes [21, 25, 30, 44].

Ideally, in a quantum world we would replace Diffie-Hellman by a protocol for which

- (a) the best classical algorithm achieves square root complexity, while
- (b) the best quantum algorithm does not provide a significant speedup.

Within the last decade isogeny-based cryptography developed as a promising candidate to provide an analogue of Diffie-Hellmann key exchange in the quantum world.

Its hardness is based on the difficulty of computing isogenies between supersingular elliptic curves. If extra information is available, like in the SIDH proposal [27], then recent breakthrough results [13, 29, 40] show a collapse of the problem’s complexity, leading to a devastating attack on the SIDH cryptosystem.

In contrast to that, in the CSIDH cryptosystem [14] no extra information is available to an attacker and the underlying isogeny computation problem remains hard. The construction is made possible by restricting to the set of supersingular elliptic curves defined over a prime field \mathbb{F}_p . This set has cardinality $N \approx \sqrt{p}$, and the best classical algorithm to recover a secret isogeny is a Meet-in-the-Middle algorithm with square root complexity $\mathcal{O}(\sqrt{N})$. However, the best quantum algorithm, due to Kuperberg [28], is subexponential in $\log N$, with complexity $2^{\mathcal{O}(\sqrt{\log N})}$.

Current CSIDH Instantiations. To guard against Kuperberg-style attacks [9, 16, 38], recent CSIDH instantiations recommend to use 512, 1024 or even 2048-bit field size for \mathbb{F}_p . To still retain highly efficient cryptosystems, current

proposals [4, 14–17, 26, 35–37] suggest to use secret keys from (sub-) sets of $\{-m, \dots, m\}^n$ of constant width m , for highly practical schemes like [16] even restricted to ternary key spaces

$$\text{SK}_1 = \{-1, 0, 1\}^n, \text{SK}_2 = \{0, 1, 2\}^n, \text{ or } \text{SK}_3 = \{-2, 0, 2\}^n.$$

Ternary keys can be guessed in 3^n steps, and the currently best

- (a) classical algorithm for recovering ternary keys is a Meet-in-the-Middle algorithm with square root time and space complexity $3^{n/2}$,
- (b) whereas the best quantum algorithm [43] is a mere quantum version of Meet-in-the-Middle, called claw-finding, providing a rather modest speedup to $3^{n/3}$.

Our Contributions. We use the Restricted Effective Group Action (REGA) framework, recently introduced in [2]. This abstraction can e.g. be instantiated via the isogeny-based CSIDH group action. Group elements are represented by vectors $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{Z}^n$, efficient implementations require to restrict the vector entries v_i to a small range $\{-m, \dots, m\}$ for some constant m . Highly efficient implementations like [16] choose ternary key spaces for \mathbf{v} .

For REGAs we introduce a REGA-DLOG problem that denotes the secret key recovery problem in REGA-based cryptography, and resembles the dlog problem for the Diffie-Hellman protocol. As a special case, REGA-DLOG_m denotes the secret key recovery problem for secret keys chosen from a small range set $\{-m, \dots, m\}^n$.

We show that the best CSIDH attacks, such as the Pollard-style algorithm going back to Galbraith-Hess-Smart [24] for smallish p and Meet-in-the-Middle (MitM) for small m , generalize to the REGA setting. For ternary key settings we show that REGA-DLOG for the key spaces $\text{SK}_1 = \{-1, 0, 1\}^n$ and $\text{SK}_2 = \{0, 1, 2\}^n$ is equivalently hard, and at least as hard as for $\text{SK}_3 = \{-2, 0, 2\}^n$. Therefore, for ternary keys it suffices to concentrate on REGA-DLOG_1 with key space SK_1 .

Since $|\text{SK}_1| = 3^n$, our MitM achieves for REGA-DLOG_1 run time $3^{0.5n}$ with memory consumption also $3^{0.5n}$. We then generalize the best time-memory CSIDH trade-off [1, 7, 16, 18] based on Parallel Collision Search (PCS), due to van Oorschot and Wiener [45] to the REGA-DLOG_1 setting resulting for memory $M \leq 3^{0.5n}$ in run time

$$T = 3^{0.75n}/M^{0.5}.$$

Notice that for maximal memory $M = 3^{0.5n}$ we again achieve MitM complexity $T = 3^{0.5n}$. However for constant M , also called the *memory-less* setting, we achieve a $T = 3^{0.75n}$ algorithm. See the dotted line (PCS) in Fig. 1 for a visualization of the interpolation between the run time exponents 0.75 and 0.5.

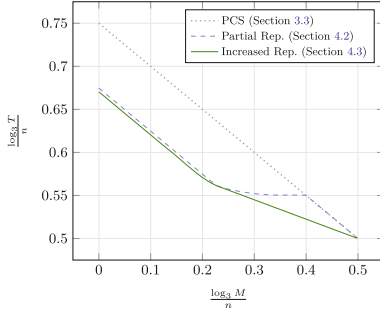


Fig. 1. Complexities for solving REGA-DLOG₁.

The REGA setting is not only a natural abstraction of isogeny-based group actions, but it also allows us —analogous to codes [6, 23, 31], lattices [25, 30, 44], and low weight discrete logarithms [22, 32]— to naturally exploit the algebraic-combinatorial benefits of using small secret keys from \mathbb{Z}^n .

Namely, by additively splitting a secret key $\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2$ with many *representations* $\mathbf{v}_1, \mathbf{v}_2 \in \{-1, 0, 1\}^n$ we significantly improve the standard PCS time-memory trade-off for $M \leq 3^{0.22n}$ to

$$3^{0.675n}/M^{0.5}.$$

Hence for memory $M = 3^{0.22n}$, which is less than the square root of MitM’s memory $3^{0.5n}$, we achieve run time $3^{0.565n}$, only slightly inferior to MitM’s time $3^{0.5n}$. In the memory-less setting, we obtain a $3^{0.675n}$ -algorithm. The tradeoff is visualized as a dashed line (Partial Rep.) in Fig. 1.

Using more elaborate representations $\mathbf{v}_1, \mathbf{v}_2 \in \{-2, \dots, 2\}^n$ of the ternary secret key, we further improve as visualized by the solid green line (Increased Rep.) in Fig. 1. Especially, we obtain a memory-less $T = 3^{0.671n}$ -algorithm, and a natural interpolation to the exponent point (0.5, 0.5) from MitM. For larger values of $m \in \{2, 3\}$ we observe that the runtime exponent c in $T = (2m + 1)^{cn}$, actually improves, we obtain for example memory-less algorithms with time $5^{0.629n}$ ($m = 2$) and $7^{0.618n}$ ($m = 3$).

Limitations of our Approach. Since all our algorithms are based on collision finding techniques, their expected run times are proven under the standard mild heuristic that the constructed functions behave like random functions with respect to collision search.

Moreover, we assume throughout the paper for sake of simplicity that a random ternary secret key $\mathbf{v} \in \{-1, 0, 1\}^n$ achieves its expected number of $n/3$ entries for each of $-1, 0$, and 1 , respectively, i.e., an equal weight distribution.

However, these limitations are no serious restrictions. First, keys with equal weight distribution have maximal entropy among all ternary keys and thus constitute the worst-case for the standard MitM algorithm (over which we improve). Second, it is not hard to see that keys with equal weight distribution amount

to a polynomial fraction of all ternary keys. And last but not least, we show that for almost all randomly chosen ternary keys one can with sub-exponential overhead always enforce an equal weight distribution.

Potential Impact of Our Representation-Based Results. Current efficient CSIDH-proposals like [16] define security levels with a memory complexity M significantly smaller than their run time complexity T . For instance [16] suggests 3 parameters sets with

$$(M_1, M_2, M_3) = (2^{80}, 2^{100}, 2^{119}) \text{ and } (T_1, T_2, T_3) = (2^{128}, 2^{128}, 2^{192})$$

for achieving NIST security level $L1, L2, L3$, respectively. The authors of [16] use a PCS-based approach for their analysis. Assuming that PCS has similar polynomial overheads as our representation method (which is certainly a complexity underestimation of the latter), for memories M_1, M_2, M_3 our representation method yields a reduced security level by 4.5, 8 and 13 bit, respectively.

Whether security bit reductions of these orders can be achieved in practice has to be validated by experiments, which are out of the scope of this work.

Organisation of our Paper. In Sect. 2 we recall the definition of Restricted Effective Group Actions (REGA) and present a REGA-based key exchange modelling CSIDH. Further we define REGA-DLOG, the main hardness problem underlying this scheme, and its small key variant REGA-DLOG $_m$. We also show that REGA-DLOG $_1$ is hardest with ternary keys from $\{-1, 0, 1\}^n$.

In Sect. 3 we generalize known cryptanalytic results such as a Pollard-style algorithm (Sect. 3.1), MitM (Sect. 3.2), and Parallel Collision Search (Sect. 3.3) to REGA-DLOG $_m$.

In Sects. 4.1 and 4.2 we introduce representation-based algorithms for REGA-DLOG $_1$, and provide a more elaborate version in Sect. 4.3. The case of keys with non-equal weight distribution is discussed in Sect. 4.4. Section 4.5 addresses REGA-DLOG $_m$ for larger $m = 2, 3$. Eventually, in Sect. 4.6 we discuss the possible practical impact of the attack.

2 Preliminaries

The Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) protocol [14] is a promising candidate for quantum-secure cryptography. Similar as its predecessor, the Couveignes-Rostovtsev-Stolbunov (CRS) scheme [19, 41], it is based on a commutative group action $\mathcal{G} \times \mathcal{X} \rightarrow \mathcal{X}$. While the underlying mathematics is quite involved, there exists a simple abstraction in the framework of *cryptographic group actions*. This framework was first introduced by Couveignes [19] under the name *hard homogenous spaces*. A more modern treatment is given in [2]. In particular the latter work also introduces *restricted effective group actions* (REGA) which model the properties of the CSIDH-based group action more closely, hence we use that framework in our analysis.

2.1 Restricted Effective Group Actions

This part follows the description of restricted effective group actions in [2] with some small modifications explained in Remark 2.2.

Definition 2.1 (Group Action). *Let (\mathcal{G}, \circ) be a group with identity element $id \in \mathcal{G}$, and \mathcal{X} a set. A map*

$$\star : \mathcal{G} \times \mathcal{X} \rightarrow \mathcal{X}$$

is a group action if it satisfies the following properties:

1. *Identity:* $id \star x = x$ for all $x \in \mathcal{X}$.
2. *Compatibility:* $(g \circ h) \star x = g \star (h \star x)$ for all $g, h \in \mathcal{G}$ and $x \in \mathcal{X}$.

Remark 2.1. In practice, one often requires that a group action is regular. This means that for any $x, y \in \mathcal{X}$ there exists precisely one $g \in \mathcal{G}$ satisfying $y = g \star x$. For instance, this is the case for the CSIDH group action which we discuss in Sect. 2.3.

Definition 2.2 (Effective Group Action). *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a group action satisfying the following properties:*

1. *The group \mathcal{G} is finite, commutative, and there exist efficient (PPT) algorithms for membership and equality testing, (random) sampling, group operation and inversion.*
2. *The set \mathcal{X} is finite and there exist efficient algorithms for membership testing and to compute a unique representation.*
3. *There exists a distinguished element $\tilde{x} \in \mathcal{X}$ with known representation.*
4. *There exists an efficient algorithm to evaluate the group action, i.e., to compute $g \star x$ given g and x .*

Then we call $\tilde{x} \in \mathcal{X}$ the origin and $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ an effective group action (EGA).

In practice, the requirements from the definition of EGA are often too strong. The limitations are reflected in the weaker notion of restricted effective group actions.

Definition 2.3 (Restricted Effective Group Action). *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a group action and let $\mathbf{g} = (g_1, \dots, g_n)$ be a set of elements in \mathcal{G} and denote $\mathcal{H} = \langle g_1, \dots, g_n \rangle$ for the subgroup generated by these elements. Assume that the following properties are satisfied:*

1. *The group \mathcal{G} is finite, commutative, and $n = \text{poly}(\log(\#\mathcal{H}))$.*
2. *The set \mathcal{X} is finite and there exist efficient algorithms for membership testing and to compute a unique representation.*
3. *There exists a distinguished element $\tilde{x} \in \mathcal{X}$ with known representation.*
4. *There exists an efficient algorithm that given $g_i \in \mathbf{g}$ and $x \in \mathcal{X}$, outputs $g_i \star x$ and $g_i^{-1} \star x$.*

Then we call $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x})$ a restricted effective group action (REGA).

Remark 2.2. Note that our definitions for EGA and REGA slightly differ from those in [2]. First, we require that the underlying group \mathcal{G} is commutative. This allows us to formulate a group action based Diffie-Hellman protocol and it is the only relevant case for our analysis. Second, we dropped the condition that the set (g_1, \dots, g_n) in the definition of REGA is a generating set for \mathcal{G} . This is necessary to include CSIDH as a possible instantiation of a REGA. In that setting, it is an open problem to determine a (compact) set of generators for the entire group \mathcal{G} . But heuristically a set of generators for a large subgroup $\mathcal{H} \subset \mathcal{G}$ is known. More details are provided in Sect. 2.3.

Vector Representation. Let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x})$ be a REGA with $\mathbf{g} = (g_1, \dots, g_n)$. Elements in \mathcal{H} can be represented as vectors $\mathbf{v} \in \mathbb{Z}^n$ under the mapping $\phi : \mathbb{Z}^n \rightarrow \mathcal{H}$, where

$$\phi : \mathbf{v} = (v_1, \dots, v_n) \mapsto \prod_{i=1}^n g_i^{v_i}.$$

Note that this representation depends on the choice of generating set \mathbf{g} for \mathcal{H} . And even fixing a set \mathbf{g} , the representation is not unique. More precisely, the kernel of the map ϕ is a lattice in \mathbb{Z}^n which is in general not known explicitly.

Via the map ϕ , we define the action of \mathbb{Z}^n on \mathcal{X} . Slightly abusing notation, we denote $\mathbf{v} \star x = \phi(\mathbf{v}) \star x$. Given a vector $\mathbf{v} \in \mathbb{Z}^n$, the action $\mathbf{v} \star x$ can be efficiently evaluated for any $x \in \mathcal{X}$ provided that the norm $\|\mathbf{v}\|$ is polynomial in $\log(\#\mathcal{H})$.

We highlight the following properties of the group action that will become important in our analysis. For any $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{Z}^n$ and $x, y \in \mathcal{X}$ it holds that

- $\mathbf{v} \star (\mathbf{u} \star x) = (\mathbf{u} + \mathbf{v}) \star x = \mathbf{u} \star (\mathbf{v} \star x)$,
- $y = (\mathbf{u} + \mathbf{v}) \star x$ implies $\mathbf{v} \star x = -\mathbf{u} \star y$,
- $x = \mathbf{v} \star (-\mathbf{v} \star x)$,
- if $\mathbf{w} \star x = (\mathbf{u} + \mathbf{v}) \star y$, then $(\mathbf{w} - \mathbf{v}) \star x = \mathbf{u} \star y$.

These properties immediately follow from the fact that $\star : \mathbb{Z}^n \times \mathcal{X} \rightarrow \mathcal{X}$ is a commutative group action.

Random Sampling. In applications, it is often required to sample elements from \mathcal{H} . If the structure of \mathcal{H} (in other words $\ker(\phi)$) is not known explicitly, then it is not possible to sample elements uniformly at random. Instead, vectors are sampled from some finite subset $S \subset \mathbb{Z}^n$. For a perfect uniform sampling, the map $\phi|_S$ would need to be bijective. In practice, one often uses $S = \{-m, \dots, m\}^n \subset \mathbb{Z}^n$ for some positive integer m . Here, m should be chosen small enough so that for two random vectors $\mathbf{v}, \mathbf{w} \in S$ the probability for $\mathbf{v} - \mathbf{w} \in \ker(\phi)$ is low. Note that this also requires that the generators g_1, \dots, g_n are evenly distributed in the group. On the other hand, if one intends to sample from a large portion of the whole group \mathcal{H} , then m must be large enough so that $\phi|_S$ is (almost) surjective. However, in some settings it is sufficient to sample elements only from a small part of the group. We already note that this is the case for the key spaces studied in our paper.

2.2 Cryptographic Group Actions and Computational Assumptions

Given an effective group action $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ one can construct a Diffie-Hellman key exchange. The setup chooses a distinguished element $x_0 \in \mathcal{X}$. Then the secret keys of Alice and Bob are group elements $g_a, g_b \in \mathcal{G}$ respectively, and the corresponding public keys are $x_a = g_a \star x_0$ and $x_b = g_b \star x_0$. Now the shared key can be computed as $x_{ab} = g_a \star x_b = g_b \star x_a$. For this protocol to be secure, the following two problems need to be hard.

1. GA-DLOG: Given $(x, y) \in \mathcal{X}^2$, determine $g \in \mathcal{G}$ such that $y = g \star x$.
2. GA-CDH: Given $(x, y, z) \in \mathcal{X}^3$, determine $w \in \mathcal{X}$ such that there exists $g \in \mathcal{G}$ with $y = g \star x$ and $w = g \star z$.

These problems are the natural generalizations of the discrete logarithm problem and the computational Diffie-Hellman problem in the classical prime-order group setting. As in [2], we refer to group actions satisfying these hardness assumptions as *cryptographic group actions*.

In the REGA setting the random sampling of group elements (i.e. secret keys) is not straightforward. A variant of the Diffie-Hellman key exchange adapted to this setting is described in Fig. 2. In essence, this is an abstract description of the CSIDH protocol introduced in [14], see also Sect. 2.3. The security of this key exchange not only relies on the hardness of GA-DLOG and GA-CDH for the group \mathcal{G} ,¹ but also on the following variant of GA-DLOG which takes into account the choice of the secret key space.

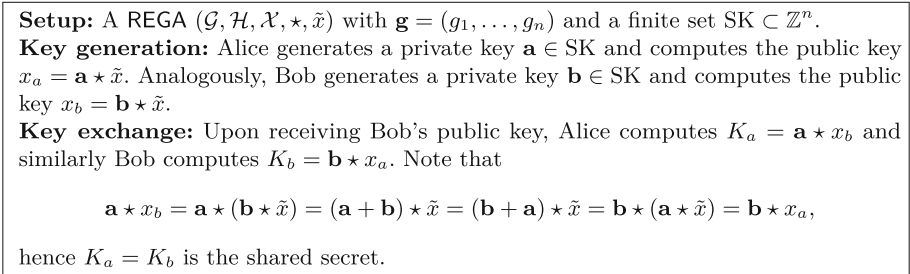


Fig. 2. A REGA-based Diffie-Hellman protocol.

Definition 2.4 (REGA-DLOG_{SK}). Let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x})$ be a REGA with $\mathbf{g} = (g_1, \dots, g_n)$ and $\text{SK} \subset \mathbb{Z}^n$ a finite subset. Given $(x, y) \in \mathcal{X}^2$, determine $\mathbf{v} \in \text{SK}$ such that $y = \mathbf{v} \star x$ if such a vector \mathbf{v} exists.

We say that the tuple $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, \mathcal{X}, y)$ is an instance of the REGA-DLOG_{SK}. In the special case where $\text{SK} = \{-m, \dots, m\}^n$ for some $m \in \mathbb{N}$, we write REGA-DLOG_m for short.

¹ More precisely, it relies on slightly modified versions of the problems, where the adversary additionally knows that there exists a solution with $g \in \mathcal{H} \subset \mathcal{G}$.

Remark 2.3. Breaking the REGA-DLOG_{SK} assumption corresponds to recovering the secret key of the REGA-based Diffie-Hellman scheme described in Fig. 2. We would like to point out that in order to break the scheme, it is sufficient to recover any (compact) vector representation of the secret key. More precisely, if $\mathbf{a} \in \text{SK}$ is Alice’s secret key and an attacker finds some $\hat{\mathbf{a}} \in \mathbb{Z}^n$ that satisfies $\phi(\hat{\mathbf{a}}) = \phi(\mathbf{a}) \in \mathcal{H}$, then he can compute the shared key as $K_{\hat{a}} = \hat{\mathbf{a}} \star x_b = \mathbf{a} \star x_b = K_a$. Of course, this further requires that the evaluation $\hat{\mathbf{a}} \star x_b$ is efficiently computable.

In the following we compare the keyspace $\text{SK} = \{-m, \dots, m\}^n$ to other choices from the literature of same cardinality. In particular the next lemma shows that it suffices to focus on the analysis of REGA-DLOG _{m} among these choices.

Lemma 2.1. *Let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x})$ be a REGA with $\mathbf{g} = (g_1, \dots, g_n)$. Let $m \in \mathbb{N}$ and consider $\text{SK}_1 = \{-m, \dots, m\}^n$, $\text{SK}_2 = \{0, \dots, 2m\}^n$, $\text{SK}_3 = \{-2m, -2(m-1), \dots, 2m\}^n$.*

1. *Then REGA-DLOG_{SK₁} and REGA-DLOG_{SK₂} are equivalent.*

Further let $\tilde{\mathcal{H}} = \{g \circ g \mid g \in \mathcal{H}\} \subset \mathcal{H}$, and $\tilde{\mathbf{g}} = (\tilde{g}_1 = g_1 \circ g_1, \dots, \tilde{g}_n = g_n \circ g_n)$.

2. *An instance $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ of REGA-DLOG_{SK₃} can be transformed to an instance $(\mathcal{G}, \tilde{\mathcal{H}}, \mathcal{X}, \star, \tilde{x}, \tilde{\mathbf{g}}, x, y)$ of REGA-DLOG_{SK₁}.*

3. *In particular if $\#\mathcal{H}$ is odd, then REGA-DLOG_{SK₃} reduces to REGA-DLOG_{SK₁}.*

Proof. Let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ be an instance of REGA-DLOG_{SK₁}. Define $y' = \mathbf{m} \star y$, where $\mathbf{m} = (m, \dots, m) \in \mathbb{Z}^n$. Then $\mathbf{w} \in \{0, \dots, 2m\}^n$ solves REGA-DLOG_{SK₂} on input $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y')$ if and only if $\mathbf{v} = \mathbf{w} - \mathbf{m}$ solves REGA-DLOG_{SK₁} on input $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$. In the same way, any instance of REGA-DLOG_{SK₂} can be transformed to an instance of REGA-DLOG_{SK₁}. This proves the first part of the lemma.

Now consider an instance $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ of REGA-DLOG_{SK₃}. Let $\tilde{\mathcal{G}}$ and $\tilde{\mathbf{g}}$ as defined in the statement of the lemma. Note that $\tilde{\mathcal{H}}$ is a subgroup of \mathcal{H} , and $\tilde{\mathbf{g}}$ is a generating set for this group. Moreover if a solution $\mathbf{v} \in \text{SK}_3$ to the REGA-DLOG_{SK₃} instance exists, then $\phi(\mathbf{v}) \in \tilde{\mathcal{H}}$. As explained in Sect. 2.1, the vector representation of group elements depends on the choice of generators. For a vector $\mathbf{v} = (v_1, \dots, v_n) \in \text{SK}_3$, we define $\tilde{\mathbf{v}} = (\frac{v_1}{2}, \dots, \frac{v_n}{2}) \in \text{SK}_1$. Then the vectors \mathbf{v} and $\tilde{\mathbf{v}}$ represent the same element in $\tilde{\mathcal{H}} \subset \mathcal{H}$ with respect to \mathbf{g} and $\tilde{\mathbf{g}}$ respectively. In other words $\prod_{i=1}^n g_i^{v_i} = \prod_{i=1}^n \tilde{g}_i^{\tilde{v}_i} \in \tilde{\mathcal{H}}$. In particular \mathbf{v} solves REGA-DLOG_{SK₃} on input $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ if and only if $\tilde{\mathbf{v}}$ solves REGA-DLOG_{SK₁} on input $(\mathcal{G}, \tilde{\mathcal{H}}, \mathcal{X}, \star, \tilde{x}, \tilde{\mathbf{g}}, x, y)$.

If $\#\mathcal{H}$ is odd, then $\tilde{\mathcal{H}} = \mathcal{H}$ and $\tilde{\mathcal{X}} = \mathcal{X}$. This observation implies the last part of the lemma. □

Remark 2.4. Note that in general, REGA-DLOG_{SK₃} and REGA-DLOG_{SK₁} are not equivalent even if $\#\mathcal{H}$ is odd. To see this, consider an instance

$(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ of REGA-DLOG_{SK₁}. Theoretically, this can be transformed to the instance $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \tilde{\mathbf{g}}, x, y)$ of REGA-DLOG_{SK₃}, where $\tilde{\mathbf{g}} = (\sqrt{g_1}, \dots, \sqrt{g_n})$. Here \sqrt{g} denotes the (unique) element in \mathcal{G} satisfying $\sqrt{g} \circ \sqrt{g} = g$. There are two issues with this transformation:

- It is not clear how to compute the elements $\sqrt{g_i}$ for $i \in \{1, \dots, n\}$.
- If the group structure is known, one can compute $\sqrt{g_i} = g_i^{(r_i+1)/2}$, where $r_i = \text{ord}(g_i)$. However the integers r_i are only bounded by $\#\mathcal{H}$, hence the evaluation of $\sqrt{g_i} \star x$ for some $x \in \mathcal{X}$ might require exponential time.

2.3 Isogeny-Based REGAs

An important instantiation of REGAs is provided by isogeny-based group actions. Here, we explain the *Commutative Supersingular Isogeny Diffie-Hellman* (CSIDH) group action.

Let p be a large prime of the form $p = 4 \cdot \ell_1 \cdots \ell_d - 1$, where the ℓ_i are small distinct odd primes. Fix the elliptic curve $E_0 : y^2 = x^3 + x$ over \mathbb{F}_p . The curve E_0 is supersingular and its \mathbb{F}_p -rational endomorphism ring is $\mathcal{O} = \mathbb{Z}[\pi]$, where π is the Frobenius endomorphism.² Let $\mathcal{E}ll_p(\mathcal{O})$ be the set of \mathbb{F}_p -isomorphism classes of elliptic curves defined over \mathbb{F}_p , with endomorphism ring \mathcal{O} . In our setting, an equivalent definition is

$$\mathcal{E}ll_p(\mathcal{O}) = \{E_A : y^2 = x^3 + Ax^2 + x \mid A \in \mathbb{F}_p \text{ and } E_A \text{ is supersingular}\}.$$

The ideal class group $cl(\mathcal{O})$ acts on the set $\mathcal{E}ll_p(\mathcal{O})$, i.e., there is a map

$$\begin{aligned} \star : cl(\mathcal{O}) \times \mathcal{E}ll_p(\mathcal{O}) &\rightarrow \mathcal{E}ll_p(\mathcal{O}) \\ ([\mathfrak{a}], E) &\mapsto [\mathfrak{a}] \star E, \end{aligned}$$

satisfying the properties from Definition 2.1 [14, Theorem 7].

The set

$$\mathbf{g} = ([l_1], \dots, [l_n]), \quad \text{where } l_i = (\ell_i, \pi - 1) \triangleleft \mathcal{O}, \quad \text{for some } n \leq d$$

generates a large subgroup $\mathcal{H} \subset cl(\mathcal{O})$. The analysis in the original CSIDH paper [14] already implies that under some heuristics $(cl(\mathcal{O}), \mathcal{H}, \mathcal{E}ll_p(\mathcal{O}), \star, E_0)$ is a REGA. We summarize the most important properties.

1. $\#cl(\mathcal{O}) \approx \#\mathcal{H} \approx \sqrt{p}$.
2. Elements in $\mathcal{E}ll_p(\mathcal{O})$ can be efficiently represented by their Montgomery coefficient $A \in \mathbb{F}_p$. Given $A \in \mathbb{F}_p$, one can efficiently test whether $E_A \in \mathcal{E}ll_p(\mathcal{O})$ using [14, Algorithm 1].
3. The distinguished element is $\tilde{x} = E_0$.
4. The expressions $[l_i] \star E$ and $[l_i]^{-1} \star E$ may be efficiently evaluated for any elliptic curve $E \in \mathcal{E}ll_p(\mathcal{O})$ and any $i \in \{1, \dots, n\}$ ([14, §3]).

² Note that we later use \mathcal{O} also in the context of standard Landau notation for complexity statements, however, its meaning will be clear from the context.

Elements of the group \mathcal{H} are represented as vectors $\mathbf{v} \in \mathbb{Z}^n$. With this notation, the CSIDH protocol corresponds precisely to the REGA-based protocol from Fig. 2. As secret keyspace SK, the original paper [14] suggests $n = d$ and $\text{SK} = \{-m, \dots, m\}^n$, where m is chosen such that $n \log(2m + 1) \approx \log(\sqrt{p})$. Hence, key recovery in CSIDH corresponds to solving REGA-DLOG $_m$. We note that here the choice of $\mathbf{g} = ([l_1], \dots, [l_n])$ guarantees that sampling from this keyspace heuristically corresponds to a close to uniform sampling in the group \mathcal{H} .

For higher security parameters (e.g., a prime field of at least 2048 bits), follow-up papers [15, 16] suggest to sample the vectors from smaller sets. For instance, it is suggested to use $n < d$ and sample vectors from

$$\text{SK}_1 = \{-1, 0, 1\}^n, \quad \text{SK}_2 = \{0, 1, 2\}^n, \quad \text{or} \quad \text{SK}_3 = \{-2, 0, 2\}^n.$$

As a consequence, the public key set $\{\mathbf{v} \star \tilde{x} : \mathbf{v} \in \text{SK}_j\}$ is only a subset of $\mathcal{E}\ell_p(\mathcal{O})$ for $j := 1, 2, 3$. Further, notice that $\#cl(\mathcal{O})$ and in particular \mathcal{G} are odd, hence Lemma 2.1 implies that the corresponding REGA-DLOG problems are equivalent for the keyspace SK $_1$ and SK $_2$, and as least as hard as for SK $_3$.

3 Adapting Techniques to the REGA-DLOG $_m$ Setting

Let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ be an instance of the REGA-DLOG $_m$ problem. Using the abstract framework of cryptographic group actions, we present different (classical) algorithms to solve this problem. These algorithms are well-known in the isogeny-based setting and have been used in the cryptanalysis of CSIDH.

In the following, let $N = \#\mathcal{H}$ (a possibly unknown) integer and $N_m = (2m + 1)^n$. In the most recent proposals for CSIDH, we are in the situation where the secret keyspace is much smaller than the group, i.e., $N_m \ll N$. In this case the best known attacks are a meet-in-the-middle (Sect. 3.2), and parallel collision search (Sect. 3.3) approach. For completeness, we also mention that there exists a memory-less Pollard-style algorithm (Sect. 3.1) with running time in $\mathcal{O}(\sqrt{N})$ which is preferable if $N_m \approx N$.

3.1 Pollard-Style Random Walks: A Galbraith-Hess-Smart Adaptation

There exists a random walk approach to find a solution $\mathbf{v} \in \mathbb{Z}^n$ (of possibly large norm) in time $\mathcal{O}(\sqrt{N})$ using only a polynomial amount of memory.

The random walks will be defined by two deterministic functions

$$f : \mathcal{X} \rightarrow \{1, \dots, n\}, \quad \sigma : \mathcal{X} \rightarrow \{-1, +1\}.$$

In the first stage of the algorithm, we set $x_0 = x$ and $\mathbf{v}_0 = \mathbf{0} \in \mathbb{Z}^n$. Then a walk of length $T \approx \sqrt{N}$ is iteratively computed as

$$x_{i+1} = g_{f(x_i)}^{\sigma(x_i)} \star x_i, \quad \mathbf{v}_{i+1} = \mathbf{v}_i + \sigma(x_i)\mathbf{e}_{f(x_i)},$$

where \mathbf{e}_i is the i -th canonical vector. The pair (x_T, v_T) is stored.

In the second stage, we set $y_0 = y$ and $\mathbf{w}_0 = \mathbf{0} \in \mathbb{Z}^n$. Then one computes

$$y_{i+1} = g_f^{\sigma(y_i)} \star y_i, \quad \mathbf{w}_{i+1} = \mathbf{w}_i + \sigma(y_i)\mathbf{e}_{f(y_i)}$$

until $y_S = x_T$ for some S . Then $(\mathbf{v}_T - \mathbf{w}_S) \star x = y$. Note that most likely $\mathbf{v}_T - \mathbf{w}_S \notin \{-m, \dots, m\}^n$, so subject to our definitions it is not a solution to REGA-DLOG. For the solution to be useful, one additionally needs a reduction algorithm red which on input $\mathbf{v} \in \mathbb{Z}^n$ computes an element $red(\mathbf{v})$ of small norm so that $red(\mathbf{v}) \star x$ can be evaluated efficiently. In isogeny based group action settings such reduction methods are available. And the corresponding Pollard-style algorithm was first described by Galbraith, Hess, and Smart [24, Section 3]. Note that for the runtime analysis it is necessary that sampling vectors of small norm in \mathbb{Z}^n corresponds to (close to) uniform sampling of group elements in \mathcal{H} as is the case for CSIDH.

3.2 Meet-in-the-Middle (MitM)

The best known attack on REGA-DLOG $_m$ is a meet-in-the-middle-attack with time and memory complexity in $\mathcal{O}(\sqrt{N_m})$. To describe the idea, we introduce the two sets

$$S_{m,0} := \{-m, \dots, m\}^{\frac{n}{2}} \times \{0\}^{\frac{n}{2}}, \quad S_{m,1} := \{0\}^{\frac{n}{2}} \times \{-m, \dots, m\}^{\frac{n}{2}}.$$

These are disjoint subsets of $S_m = \{-m, \dots, m\}^n$ of size $\sqrt{N_m}$ each. Moreover, any element $\mathbf{v} \in S_m$ has a unique representation as $\mathbf{v}_0 + \mathbf{v}_1$ with $\mathbf{v}_0 \in S_{m,0}$ and $\mathbf{v}_1 \in S_{m,1}$. So given two set elements $x, y \in \mathcal{X}$, the problem of finding $\mathbf{v} \in S_m$ with $y = \mathbf{v} \star x$ reduces to finding vectors $\mathbf{v}_0 \in S_{m,0}$ and $\mathbf{v}_1 \in S_{m,1}$ with

$$\mathbf{v}_0 \star x = (-\mathbf{v}_1) \star y. \tag{1}$$

The time T and memory complexity M of this procedure are linear in the size of the subsets $|S_{m,0}| = |S_{m,1}|$, and therefore gives $T = M = \mathcal{O}(\sqrt{N_m})$. Concretely, for \mathbf{v} being chosen from a ternary alphabet we have $T = M = \mathcal{O}(3^{\frac{n}{2}})$.

In practical applications the memory requirements of the MitM approach usually render it ineffective and require to resort to time-memory trade-offs. The naive trade-off for the MitM algorithm given W units of memory processes the subset $S_{m,0}$ in batches of size W . For each batch it iterates through all candidates $\mathbf{x}_1 \in S_{m,1}$ for \mathbf{v}_1 and checks for a match in the current batch. If no match is found it continues with the next batch. Straightforward analysis shows that this reduces the memory to $\tilde{\mathcal{O}}(W)$, while increasing the time complexity to $\tilde{\mathcal{O}}(N_m/W)$.

However, in the limited memory setting the Parallel Collision Search (PCS) technique by van Oorschot and Wiener is known to offer a better trade-off behavior.

3.3 Parallel Collision Search (PCS)

PCS is a technique to accelerate the search for multiple collisions between two functions f_0 and f_1 by the use of memory. A single collision between functions f_0, f_1 with domain D can be found in time $\tilde{O}(\sqrt{|D|})$ using a polynomial amount of memory by standard techniques. The PCS algorithm now allows to find W collisions in time $\tilde{O}(\sqrt{|D| \cdot W})$ using $\tilde{O}(W)$ memory. This yields a $\tilde{O}(\sqrt{W})$ speedup over naive repetition of the memory-less procedure. We formalize this in the following lemma.

Lemma 3.1 (Parallel Collision Search). *Let $f_i: D_i \rightarrow D$ with $|D_i| = |D|$, $i = 0, 1$ be two random functions that can be evaluated in time polynomial in $\log D$. Then there is an algorithm that returns W collisions between f_0 and f_1 in time $T = \tilde{O}(\sqrt{|D| \cdot W})$ using $M = \tilde{O}(W)$ memory.*

Here we do not want to dive into the details on how the technique achieves the acceleration, for those details the reader is referred to [45]. Instead we want to focus on its application to the REGA-DLOG _{m} or more specifically to the CSIDH case. Therefore we first reformulate the search for \mathbf{v}_0 and \mathbf{v}_1 as a collision search procedure. Let $S_m^{n/2} := \{-m, \dots, m\}^{\frac{n}{2}}$ and $H: \{0, 1\}^* \rightarrow S_m^{n/2}$ be a hash function. Further, define the functions $f_i: S_{m,i} \rightarrow S_m^{n/2}$, $i = 0, 1$ as

$$f_0: \mathbf{v} \mapsto H(\mathbf{v} \star x) \quad \text{and} \quad f_1: \mathbf{v} \mapsto H((-\mathbf{v}) \star y). \tag{2}$$

Now clearly \mathbf{v}_0 and \mathbf{v}_1 form a collision between f_0 and f_1 (compare to Eq. (1)). However, not every collision $(\mathbf{x}_0, \mathbf{x}_1)$ between f_0 and f_1 leads to \mathbf{v} , as the collision might only be a collision in the hash function H , but not necessarily implying that $\mathbf{x}_0 \star x = (-\mathbf{x}_1) \star y$. In order to find the single distinguished (often called *golden*) collision $(\mathbf{v}_0, \mathbf{v}_1)$ that leads to \mathbf{v} we, therefore, have to find all collisions between f_0 and f_1 . Now, instead of naively applying the standard memory-less collision search multiple times we make use of PCS to find W collisions at a time using W units of memory. We outline this procedure in pseudocode in Algorithm 1.

Algorithm 1: PCS-TRADEOFF TO SOLVE REGA-DLOG _{m}

Input : Functions $f_i: D_i \rightarrow D$, $i = 0, 1$ with $|D_i| = |D|$, W units of memory, instance $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ of the REGA-DLOG _{m}

Output: solution \mathbf{v} to the REGA-DLOG _{m} instance (x, y) satisfying $y = \mathbf{v} \star x$

- 1 **repeat**
 - 2 | find W collisions $(\mathbf{w}_i, \mathbf{z}_i)$ between f_0, f_1 using PCS
 - 3 **until** $\exists j: y = (\mathbf{w}_j + \mathbf{z}_j) \star x$
 - 4 **return** $\mathbf{w}_j + \mathbf{z}_j$
-

Analysis. Let us briefly analyze the correctness of the procedure. For the functions f_0, f_1 defined in Eq. (2), we have already shown that the pair of inputs $(\mathbf{v}_0, \mathbf{v}_1)$, with $\mathbf{v} = \mathbf{v}_0 + \mathbf{v}_1$ forms a collision. Therefore the algorithm can succeed in recovering $\mathbf{v} = \mathbf{v}_0 + \mathbf{v}_1$ by finding random collisions between those functions.

Next let us analyze the running time. As already observed, we need to recover all collisions between the functions to guarantee to find the distinguished collision that leads to \mathbf{v} . Further, we expect a total amount of $C = |D| = \sqrt{N_m}$ collisions between f_0 and f_1 . Therefore after $\text{poly}(n) \cdot \sqrt{N_m}/W$ applications of the PCS technique, each yielding W collisions, we gathered a total of $\text{poly}(n) \cdot \sqrt{N_m}$ collisions. Under a standard assumption that treats those collisions as randomly sampled from the set of all collisions, we found each collision between f_0 and f_1 with high probability using a standard coupon collectors argument. This implies especially that we found the distinguished collision $(\mathbf{v}_0, \mathbf{v}_1)$ and the algorithm terminates. Each of the $\tilde{O}(\sqrt{N_m}/W)$ comes at a cost of $\tilde{O}(\sqrt{\sqrt{N_m} \cdot W})$ (compare to Lemma 3.1), yielding a running time of

$$T_{\text{PCS}} = \tilde{O}\left(\frac{(N_m)^{\frac{3}{4}}}{\sqrt{W}}\right),$$

while the memory complexity is given as $M = \tilde{O}(W)$.

4 A New Time-Memory Trade-Off Using Representations

In the following we make use of the representation technique to improve the time-memory trade-off behavior of the PCS technique in the REGA setting. Therefore we first re-define the used functions to use larger domains. At first sight, this comes at the downside of increasing the cost for the collision search procedure. However, by carefully choosing the new domains we guarantee that there are several collisions $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, \dots, N$ that allow to recover the secret \mathbf{v} . In turn it is not necessary to compute all existing collisions but only a $1/N$ -fraction to find one of these distinguished collisions and recover \mathbf{v} , which overall results in a runtime advantage. Motivated by recent proposals to use ternary key spaces and for didactic reasons we first concentrate on the case of $m = 1$. Moreover, in Sects. 4.1 to 4.3, we assume that the solution to REGA-DLOG₁ has the same number of (-1) -, 0 -, and 1 -entries. Generalizations to the case of arbitrary weight distribution and arbitrary m are given in Sect. 4.4 and Sect. 4.5, respectively.

4.1 A First Representation-Based Approach

We start with a (slightly) sub-optimal variant of our algorithm for didactic reasons. In the following sections we then subsequently refine this initial algorithm.

Let the set of ternary vectors of length n with exactly $\alpha n \pm 1$ entries each be defined as

$$\mathcal{T}^n(\alpha) := \{\mathbf{x} \in \{-1, 0, 1\}^n \mid \mathbf{x} \text{ contains exactly } \alpha n \text{ (+1) and } \alpha n \text{ (-1) entries}\}.$$

Now we start by redefining the functions over different domains as

$$f_0, f_1: \mathcal{T}^n(\alpha) \rightarrow \mathcal{T}^n(\alpha). \tag{3}$$

Apart from this the functions remain as specified in Eq. (2), where the hash function is now defined on $\mathbf{H}: \{0, 1\}^* \rightarrow \mathcal{T}^n(\alpha)$ and $\alpha \in \llbracket 0, 1 \rrbracket$ is an optimization parameter.

Our algorithm now again searches for collisions between f_0, f_1 via the PCS strategy, until a collision $(\mathbf{x}_0, \mathbf{x}_1)$ with $\mathbf{x}_0 + \mathbf{x}_1 = \mathbf{v}$ is found. A pseudocode description is obtained by using the re-defined functions together with $m = 1$ as input for Algorithm 1.

Analysis. Recall that a collision $(\mathbf{x}_0, \mathbf{x}_1)$ in f is either caused by a collision in the hash function, i.e., $\mathbf{x}_0 \star x \neq (-\mathbf{x}_1) \star y$, but $\mathbf{H}(\mathbf{x}_0 \star x) = \mathbf{H}((-\mathbf{x}_1) \star y)$ or we have

$$\mathbf{x}_0 \star x = (-\mathbf{x}_1) \star y \iff (\mathbf{x}_0 + \mathbf{x}_1) \star x = y,$$

In the latter case we call the collision *real* and conclude that $\mathbf{x}_0 + \mathbf{x}_1 = \mathbf{v}$, since \mathbf{v} is sufficiently unique. This implies that any *real* collision leads to recovering \mathbf{v} .

Next, let us analyze the amount of real collisions $(\mathbf{x}_0, \mathbf{x}_1)$. For this it suffices to analyze the amount of $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{T}^n(\alpha)$ which satisfy $\mathbf{x}_0 + \mathbf{x}_1 = \mathbf{v}$. Those pairs $(\mathbf{x}_0, \mathbf{x}_1)$ are usually called *representations* of \mathbf{v} . Note that the amount of these representations of $\mathbf{v} \in \mathcal{T}^n(1/3)$ is

$$R = \binom{n/3}{n/6}^2 \binom{n/3}{\varepsilon, \varepsilon, n/3 - 2\varepsilon},$$

where $\varepsilon = (\alpha - \frac{1}{6})n$. Here the binomial coefficient counts the possibilities how $\frac{n}{6}$ of the 1 (resp. -1) entries of \mathbf{v} can be contributed from \mathbf{x}_0 , while the remaining $\frac{n}{6} - 1$ (resp. -1) entries have to be present in \mathbf{x}_1 . The multinomial coefficient then counts the possibilities how the remaining 1s and -1 s can cancel out to represent the 0s in \mathbf{v} . Since our choice of α will ensure $R \geq 1$ the algorithm can succeed in recovering \mathbf{v} by sampling random collisions between f_0 and f_1 .

Let us now analyze the time complexity. We expect that after computing $\frac{C}{R}$ random collisions we encounter one that forms a representation of \mathbf{v} , where C is the total amount of existing collisions. Again we expect a total number of $C = |\mathcal{T}^n(\alpha)|$ collisions. Further, under the standard assumption that the functions still behave like random functions with respect to collision search, a single collision can be found in time

$$T_1 := \tilde{O}\left(\sqrt{|T^n(\alpha)|}\right) = \tilde{O}\left(\binom{n}{\alpha n, \alpha n, (1-2\alpha)n}\right)^{\frac{1}{2}}$$

and using Lemma 3.1 we can find W collisions in time $T_W = \sqrt{W} \cdot T_1$ using $M = \tilde{O}(W)$ memory. Computing the required $\frac{C}{R}$ collisions using $M = \tilde{O}(W)$ memory therefore takes expected time

$$T = \tilde{O}\left(\frac{C}{R \cdot W} \cdot T_W\right) = \tilde{O}\left(\frac{|T^n(\alpha)|^{\frac{3}{2}}}{R \cdot \sqrt{W}}\right),$$

as long as $\frac{C}{R} \geq W$.

To obtain a running time of the form $T = \tilde{O}(3^{c(\alpha)n})$ we approximate the binomial and multinomial coefficients in T using the well known approximation

$$\binom{n}{k} = \tilde{O}\left(2^{nH(k/n)}\right), \tag{4}$$

where $H(x) := -x \log_2(x) - (1-x) \log_2(1-x)$ denotes the binary entropy function. We then perform a numerical optimization using the *python* library *scipy* to find the optimal α for a given amount of memory $W = 3^{\omega n}$, $\omega \in [0, 0.5]$. We apply this strategy for all our representation based algorithms and make our optimization code open source.³ The way we access the numerical optimization framework is inspired by the code of Bonnetain, Bricout, Schrottenloher and Shen [8].

We illustrate the obtained runtime exponent as a function of the available memory in Fig. 3 and give as comparison the standard PCS exponent and the naive MitM trade-off. For an available memory that is only polynomial in n , i.e., $\omega = 0$ we improve the running time from $\tilde{O}(3^{0.75n})$ to $3^{0.675n}$. In turn this leads to an improved trade-off with time complexity $T = 3^{0.675n}/M^{0.5}$ for any available memory $M \leq 3^{0.22n}$. From there on the optimal choice of α does not fulfill the constraint $\frac{C}{R} \geq W$. However, by slightly adapting the choice of α it is possible to enforce $\frac{C}{R} \geq W$ up to $W < 3^{0.265n}$. From there on more memory does not translate into a runtime advantage, as indicated by the horizontal dotted line.

4.2 Interpolation Using Partial Representations

In order to interpolate between the standard PCS technique and our representation based method from the previous section we adapt in the following the concept of partial representations introduced independently in [12, 22] to our setting. In turn this allows us to achieve runtime improvements for $W \geq 3^{0.265n}$.

³ <https://github.com/Memphis/Low-Memory-Attacks-on-Small-Key-CSIDH>.

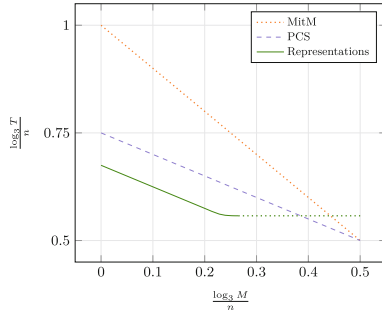


Fig. 3. Complexity of PCS, MitM and the representation-based trade-off

So far both methods – standard PCS as well as our representation approach – split the secret \mathbf{v} in the sum of two vectors \mathbf{x}_0 and \mathbf{x}_1 . For the standard PCS technique the vectors \mathbf{x}_0 and \mathbf{x}_1 have disjoint support, while for the representation method the support overlaps. Partial representations now combine both cases by introducing an additional optimization parameter $\delta \in \llbracket 0, 1 \rrbracket$ that defines how big the fraction of overlapping support of both vectors is. More precisely the secret $\mathbf{v} \in \{-1, 0, 1\}^n$ is split as

$$\mathbf{v} = \underbrace{(\mathbf{y}_0, \mathbf{0}, \mathbf{z}_0)}_{\mathbf{x}_0} + \underbrace{(\mathbf{0}, \mathbf{y}_1, \mathbf{z}_1)}_{\mathbf{x}_1} = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{z}_0 + \mathbf{z}_1)$$

with $\mathbf{y}_0, \mathbf{y}_1 \in \{-1, 0, 1\}^{\frac{(1-\delta)n}{2}}$ and $\mathbf{z}_0, \mathbf{z}_1 \in \mathcal{T}^{\delta n}(\alpha)$, where α is again an optimization parameter. That means the vectors \mathbf{x}_0 and \mathbf{x}_1 have disjoint support on the first $(1 - \delta)n$ coordinates, while on the last δn their support overlaps.

Let us now re-define the functions f_0, f_1 according to partial representations. Therefore we use the following sets as domains

$$\begin{aligned} D_0 &:= \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \{0\}^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\delta n}(\alpha) \quad \text{and} \\ D_1 &:= \{0\}^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha), \end{aligned} \tag{5}$$

and define the common image space as $D := \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha)$. Leading to the functions

$$f_i: D_i \rightarrow D, \quad i = 0, 1. \tag{6}$$

Again the concrete definition of the functions remains as given in Eq. (2), where we now require a hash function $\mathbf{H}: \{0, 1\}^* \rightarrow D$.

In the following we use Algorithm 1 with our adapted functions from Eq. (6) and $m = 1$.

Analysis. The correctness follows from the analysis of the previous section and the fact that our choice of parameters will ensure that there is at least one *real* collision, i.e. a representation of the solution or following the notation of the previous section $R \geq 1$.

Let us, hence, start by analyzing the, now changed, amount of representations R . Note, that on the first $(1 - \delta)n$ coordinates, where elements from D_0 and D_1 have disjoint support, we have only a single possible decomposition of any element in $\mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3)$. Therefore we assume that the solution \mathbf{v} lies in

$$\mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(1/3),$$

meaning the 1 and -1 entries distribute according to their expectation proportionally onto the three segments of length $\frac{(1-\delta)n}{2}$, $\frac{(1-\delta)n}{2}$ and δn . However, in the following we show that ensuring such a distribution of the coordinates causes at most a polynomial overhead.

Note that the probability over the random choice of $\mathbf{v} \in \mathcal{T}^n(1/3)$ for \mathbf{v} having a proportional coordinate distribution over the three segments is

$$\frac{\binom{\frac{(1-\delta)n}{2}}{\frac{(1-\delta)n}{6}, \frac{(1-\delta)n}{6}, \frac{(1-\delta)n}{6}}^2 \binom{\delta n}{\delta n/3, \delta n/3, \delta n/3}}{\binom{n}{n/3, n/3, n/3}} = \frac{1}{\text{poly}(n)},$$

which follows from approximating the binomial coefficients via Eq. (4). Note that by randomly permuting the order of the generators of \mathcal{G} we can obtain independent uniform distributions of the 1 and -1 entries on \mathbf{v} , each having a probability of $\frac{1}{\text{poly}(n)}$ to distribute the coordinates as required. Therefore we expect $\text{poly}(n)$ repetitions of the algorithm with random permutations of the generators to ensure this distribution in at least one of the executions.

On the last δn coordinates of elements from $\mathbf{x}_i \in D_i$, we obtain multiple representations of $\mathbf{v} = \mathbf{x}_0 + \mathbf{x}_1$ as sum of two elements. Similar to the analysis in the previous section the amount of such representations of one element from $\mathcal{T}^{\delta n}(1/3)$ as the sum of two elements from $\mathcal{T}^{\delta n}(\alpha)$ is given as

$$R = \binom{\delta n/3}{\delta n/6}^2 \binom{\delta n/3}{\varepsilon, \varepsilon, \delta n/3 - 2\varepsilon},$$

where $\varepsilon = (\alpha - \frac{1}{6})\delta n$.

The complexity analysis follows along the lines of the analysis in Sect. 4.1 with the difference that a single collision search now comes at the cost of

$$T_1 = \tilde{\mathcal{O}}(\sqrt{|D|}) = \tilde{\mathcal{O}}\left(\left(\left(\binom{\frac{(1-\delta)n}{2}}{\frac{(1-\delta)n}{6}, \frac{(1-\delta)n}{6}, \frac{(1-\delta)n}{6}}\right) \binom{\delta n}{\alpha\delta n, \alpha\delta n, (1-2\alpha)\delta n}\right)\right)^{\frac{1}{2}}.$$

The final complexity is then analogously given as $T = \tilde{\mathcal{O}}(\frac{C}{R \cdot W} \cdot T_W) = \tilde{\mathcal{O}}\left(\frac{|D|^{\frac{3}{2}}}{R \cdot \sqrt{W}}\right)$, as long as $\frac{C}{R} \geq W$, where still $T_W = \sqrt{W} \cdot T_1$. The memory complexity is still dominated by the application of the PCS with $M = \tilde{\mathcal{O}}(W)$.

In Fig. 4 we illustrate the asymptotic running time exponent obtained by numerical optimization of α, δ as a function of the memory. We observe that

partial representations enable a smooth interpolation between the representation method from Sect. 4.1 and the PCS technique (Sect. 3.3), while providing improvements over both methods for any $3^{0.25n} \leq M < 3^{0.4n}$.

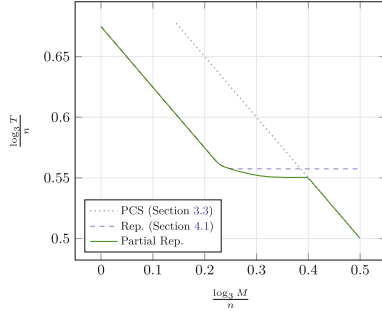


Fig. 4. Complexity of PCS, the representation trade-off, and partial representations.

4.3 Increasing the Amount of Representations

In the following we again slightly adapt the domains of the functions to include elements with coordinates in $\{-2, \dots, 2\}$ rather than $\{-1, 0, 1\}$. While, as before, this increases the size of the domains and, hence, the time for the collision search, it also yields an increased amount of representations, leading to a runtime improvement.

Note that in terms of representations any -1 can additionally be represented as $-2 + 1$ (resp. $1 + (-2)$), accordingly any 1 as $-1 + 2$ (resp. $2 + (-1)$) and any 0 as $-2 + 2$ (resp. $2 + (-2)$). Let the set of vectors with $\alpha n \pm 1$ entries each and $\beta \pm 2$ entries each be denoted as

$$\mathcal{T}^n(\alpha, \beta) := \{\mathbf{x} \in \{-2, \dots, 2\}^n \mid |\mathbf{x}|_1 = |\mathbf{x}|_{-1} = \alpha n \wedge |\mathbf{x}|_2 = |\mathbf{x}|_{-2} = \beta n\}, \quad (7)$$

where $|\mathbf{x}|_i = |\{j \in \{1, \dots, n\} \mid x_j = i\}|$.

We now adapt the domains of the previous section, i.e., we still use partial representations, but now also including -2 and 2 entries. Therefore let the new domains be

$$\begin{aligned} \tilde{D}_0 &:= \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \{0\}^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\delta n}(\alpha, \beta) \quad \text{and} \\ \tilde{D}_1 &:= \{0\}^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha, \beta), \end{aligned} \quad (8)$$

and re-define the common image space as $\tilde{D} := \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha, \beta)$, where $\delta \in \llbracket 0, 1 \rrbracket$ is subject to optimization and α, β are determined later. The functions are then defined over

$$f_i: \tilde{D}_i \rightarrow \tilde{D}, \quad i = 0, 1, \quad (9)$$

with their precise mapping still as given in Eq. (2), requiring now a hash function $H: \{0, 1\}^* \rightarrow \tilde{D}$.

We now analyze the complexity of Algorithm 1 with input $m = 1$ and functions as specified in Eq. (9).

Analysis. The analysis again follows along the lines of the analysis of the previous section with the main difference lying in the amount of representations R and the now increased domain size $|\tilde{D}|$. Let us start by examining the amount of representations R . We, again, assume \mathbf{v} to be from $\mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(1/3)$, which we ensure by random permutations of the generators leading to polynomial overhead. In turn there exists exactly one decomposition of \mathbf{v} in the sum of two elements from \tilde{D}_1 and \tilde{D}_2 with respect to the the first $(1-\delta)n$ coordinates. Multiple representations only exist for the last δn coordinates. We have the following possibilities to represent a $-1, 0$ and 1 entry in $\mathbf{v} = \mathbf{x}_0 + \mathbf{x}_1$

$$\begin{aligned}
 0 : & \quad \underbrace{0+0}_{z_0}, & \underbrace{1-1}_{z_1}, & \underbrace{-1+1}_{z_1}, & \underbrace{2-2}_{z_2}, & \underbrace{-2+2}_{z_2}, \\
 1 : & \quad \underbrace{1+0}_{\frac{\delta n}{6}-o}, & \underbrace{0+1}_{\frac{\delta n}{6}-o}, & \underbrace{2-1}_o, & \underbrace{-1+2}_o, & \\
 -1 : & \quad \underbrace{-1+0}_{\frac{\delta n}{6}-o}, & \underbrace{0-1}_{\frac{\delta n}{6}-o}, & \underbrace{-2+1}_o, & \underbrace{1-2}_o. &
 \end{aligned} \tag{10}$$

Here the variable below each of the representations specifies how often we want to use the corresponding representation to represent a corresponding coordinate of \mathbf{v} . For example we expect z_0 many of the 0 entries in \mathbf{v} to be represented in the sum $\mathbf{v} = \mathbf{x}_0 + \mathbf{x}_1$ as $0+0$, z_1 as $1-1$, z_1 as $-1+1$, z_2 as $2-2$ and z_2 as $-2+2$. It follows that we need to ensure

$$z_0 + 2z_1 + 2z_2 = \frac{\delta n}{3} \quad \Leftrightarrow \quad z_0 = \frac{\delta n}{3} - 2z_1 - 2z_2,$$

as in total we need to represent $\frac{\delta n}{3}$ zeros of \mathbf{v} . Note that the total amount of 1 (resp. -1) entries sums to $\frac{\delta n}{6} - o + \frac{\delta n}{6} - o + o + o = \frac{\delta n}{3}$ as required (since there are that many -1 and 1 entries in the last δn coordinates of \mathbf{v}). Note that the parameters z_1, z_2 and o are optimization parameters of the algorithm. Given the proportions specified in Eq. (10), we can directly derive the amount of representations as

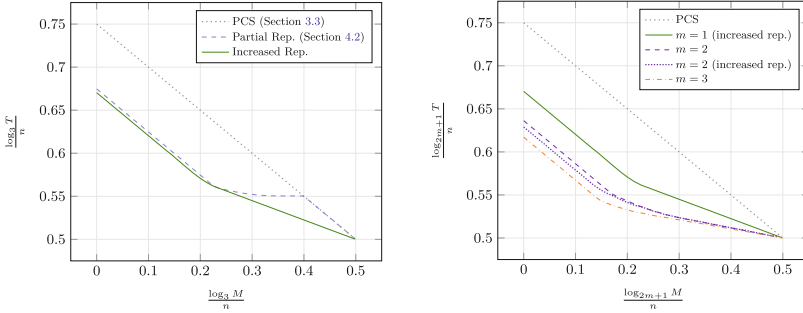
$$R = \binom{\frac{\delta n}{3}}{z_0, z_1, z_1, z_2, z_2} \binom{\frac{\delta n}{3}}{\frac{\delta n}{6} - o, \frac{\delta n}{6} - o, o, o}^2,$$

where the first term counts the possibilities to represent 0s and the second the representations of ± 1 entries. A simple counting of the representations from Eq. (10) including a ± 1 or ± 2 yields that the initial domains \tilde{D}_1, \tilde{D}_2 need to satisfy

$$\alpha = \frac{1}{6} + \frac{z_1}{\delta} \quad \text{and} \quad \beta = \frac{z_2 + o}{\delta}.$$

From here the analysis is identical to the one from Sect. 4.2, leading to a time complexity of $T = \tilde{\mathcal{O}}\left(\frac{C}{R \cdot W} \cdot T_W\right) = \tilde{\mathcal{O}}\left(\frac{|\tilde{D}|^{\frac{3}{2}}}{R \cdot \sqrt{W}}\right)$, as long as $\frac{|\tilde{D}|}{R} \geq W$, and memory complexity $M = \tilde{\mathcal{O}}(W)$.

In Fig. 5a we illustrate the obtained runtime exponent. We observe that the increased amount of representations allows to naturally connect the trade-off to the (0.5, 0.5) endpoint of MitM.



(a) Complexity of different approaches. (b) Complexity for different choices of m .

Fig. 5. On the left: Comparison of different representation based methods. On the right: Comparison of representation based methods for different m .

4.4 Enforcing an Equal Weight Distribution

Recall that in previous sections we always assumed for simplicity that we attack ternary vectors with equally balanced (up to rounding) number of (-1) -, 0 -, and 1 -entries. We now show that for almost all ternary vectors we can enforce such an equal weight distribution by increasing the dimension of the REGA-DLOG₁-problem from n to $n + \mathcal{O}(\sqrt{n})$. Our argument extends to all REGA-DLOG _{m} with constant m .

Notice that our algorithms are of complexity $T = 3^{cn}$ for some constant c , and thus fully exponential in the dimension n . Therefore, our dimension increase only leads to a subexponential overhead $3^{\mathcal{O}(\sqrt{n})}$, i.e., we achieve asymptotic run time

$$3^{c(n+\mathcal{O}(\sqrt{n}))} = T \cdot 3^{\mathcal{O}(\sqrt{n})} = 3^{cn(1+o(1))}.$$

Idea of Balancing. Let \mathbf{v} be a random ternary, and denote by $n_i, i \in \{-1, 0, 1\}$ its numbers of i -entries. Since $n_i \leq n$, we can guess all n_i in polynomial time $\mathcal{O}(n^2)$. We show that with high probability all n_i are bounded by $n/3 \pm \mathcal{O}(\sqrt{n})$. Without loss of generality, let n_{-1} be the maximal value. We then add $n_{-1} - n_0$ coordinates for 0 -entries, and $n_{-1} - n_1$ coordinates for 1 -entries. These are in total $\ell = \mathcal{O}(\sqrt{n})$ coordinates.

To this end, let $(\mathcal{G}, \mathcal{H}, \mathcal{X}, \star, \tilde{x}, \mathbf{g}, x, y)$ be an instance of the REGA-DLOG₁ problem with $\mathbf{g} = (g_1, \dots, g_n)$ and a ternary solution \mathbf{v} satisfying $\mathbf{v} \star x = y$.

Let id be the neutral element in \mathcal{G} , and let $\mathbf{g}' = (g_1, \dots, g_n, id, \dots, id)$ be the set of generators enhanced by ℓ times id . Then any $\mathbf{u} = (\mathbf{v}, \mathbf{w})$ with $\mathbf{w} \in \mathbb{Z}^\ell$ is a solution for the dimension-increased instance with \mathbf{g}' iff \mathbf{v} is a solution for the original instance with \mathbf{g} . Especially, we obtain a solution for our n -dimensional

instance by solving the $n + \ell = n + \mathcal{O}(\sqrt{n})$ -dimension instance, and cutting off the last ℓ coordinates.

Chernoff Argument. It remains to show that all n_i differ from $n/3$ by at most $\mathcal{O}(\sqrt{n})$. Since \mathbf{v} is a random ternary vector, all n_i are binomially distributed random variables with $\mathbb{E}[n_i] = n/3$. We use the Chernoff bound

$$\Pr [|n_i - \mathbb{E}[n_i]| \geq \delta \mathbb{E}[n_i]] \leq 2e^{-\mathbb{E}[n_i]\delta^2/3} \text{ for } 0 < \delta < 1.$$

Define $\delta = \frac{3c}{\sqrt{n}}$ for some constant c . Then $\Pr[|n_i - n/3| \geq c\sqrt{n}] \leq 2e^{-c^2}$. Thus, for sufficiently large c , almost all ternary vectors reach their expected value $n/3$ up to an $\mathcal{O}(\sqrt{n})$ error term.

4.5 The Case of Arbitrary m

Intuitively it is clear that a similar approach as in Sects. 4.1 to 4.3 can be taken to solve the REGA-DLOG $_m$ for an arbitrary integer m . One simply defines the functions over appropriate domains, which allow for multiple representations of $\mathbf{v} \in \{-m, \dots, m\}^n$ and then applies Algorithm 1 with those functions and the respective choice of m . The main obstacle with this approach lies in the computation of the representations, which already for $m = 1$ became quite technical if applying the technique to its full extend (compare to Sect. 4.3).

However, for completeness we specify in the following the running time for the case of general m in dependence on the domain size and the amount of representations. The result is an immediate implication of our previous analysis.

Let the functions be specified as $f_i: S_i \rightarrow S, i = 0, 1$, with the mapping as defined in Eq. (2), where $H: \{0, 1\}^* \rightarrow S$ and $|S_0| = |S_1| = |S|$. Furthermore, let every element $\mathbf{v} \in \{-m, \dots, m\}^n$ have R representations as the sum of elements from S_0, S_1 , i.e., there are R different pairs $(\mathbf{x}_0, \mathbf{x}_1) \in S_0 \times S_1$ with $\mathbf{v} = \mathbf{x}_0 + \mathbf{x}_1$.

Then v can be found via Algorithm 1 with functions f_0, f_1 in time $T = \tilde{\mathcal{O}}\left(\frac{|S|^{\frac{3}{2}}}{R \cdot \sqrt{W}}\right)$, as long as $\frac{|S|}{R} \geq W$, using memory $M = \tilde{\mathcal{O}}(W)$.

We additionally computed the running time of our technique for $m \in \{2, 3\}$ for an appropriate choice of function domains. We illustrate the corresponding runtime exponents in Fig. 5b.

For obtaining the running times we used in the case of $m = 2$ addends $\mathbf{x}_0, \mathbf{x}_1 \in \{-2, \dots, 2\}$ ($m = 2$) and $\mathbf{x}_0, \mathbf{x}_1 \in \{-3, \dots, 3\}$ ($m = 2$ (increased rep.)) to represent the solution $\mathbf{v} = \mathbf{x}_0 + \mathbf{x}_1$. In the case of $m = 3$ we used only addends $\mathbf{x}_0, \mathbf{x}_1 \in \{-3, \dots, 3\}$. For the full technical details of the analysis the reader is referred to Appendix A. It can be observed, that for increasing m the runtime exponent in dependence on search space improves. However, since the improvement is getting smaller for growing m we conjecture that the exponent converges.

4.6 Potential Impact on Bit Security Level

In this section we approximate the maximal bit security reduction for suggested parameter sets for CSIDH by the representation method. In our comparison we assume that the standard PCS based time-memory trade-off (compare to Sect. 3.3) suffers the same polynomial overhead as the representation based approach. Since this might underestimate the overhead of the representation based trade-off, the numbers should be seen as a maximal potential gain. Practical experiments will have to determine to which extend this gain can be realized in practice.

In [16] three concrete parameter instantiations for ternary-key CSIDH are given, respectively aiming at satisfying NIST security level L_1 , L_2 and L_3 . For matching the security definition of category L_i the authors impose restrictions on the memory and time complexity of $M_i = 2^{w_i}$ and $T_i = w^{t_i}$ with

$$(w_1, w_2, w_3) = (80, 100, 119) \quad \text{and} \quad (t_1, t_2, t_3) = (128, 128, 192).$$

In order to match those security definitions a number of generators n_i equal to $n_1 = 139$ for L_1 , $n_2 = 148$ for L_2 and $n_3 = 210$ for L_3 is proposed. The security of those parameter sets is determined via the PCS time-memory trade-off.

In the memory restriction the authors conservatively ignore polynomial factors, i.e., it holds $M_i = 3^{c_i n_i} = 2^{w_i}$, which allows to determine the asymptotic memory exponent as $c_i = \frac{w_i}{n_i \cdot \log_2 3}$. For example for $i = 1$ we obtain $c_1 \approx 0.3631$, which yields an asymptotic running time of the PCS approach of $T_{\text{PCS}} = 3^{0.5685n}$. In comparison our technique improves the running time to $T_{\text{Rep}} = 3^{0.5316n}$, corresponding to a gain of

$$\frac{T_{\text{PCS}}}{T_{\text{Rep}}} = 3^{0.5685n} = 3^{0.0369n},$$

which for $n_1 = 139$ yields a reduced security level by $0.0369 \cdot n_1 \cdot \log_2 3 \approx 8.13$ bit.

A similar analysis for the cases of $i = 2$ yields $c_2 \approx 0.4263$ with $T_{\text{PCS}} = 3^{0.5369n}$ and $T_{\text{Rep}} = 3^{0.5174n}$ corresponding to a gain of 4.57 bit. The case of $i = 3$ yields $c_3 \approx 0.3575$ with $T_{\text{PCS}} = 3^{0.5713n}$ and $T_{\text{Rep}} = 3^{0.5330n}$ reducing the security level by 12.75 bit.

Acknowledgements. Sabrina Kunzweiler and Alexander May were funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

A The Case of Larger m

For larger choices of m we still assume that each coordinate is present $\frac{n}{2m+1}$ times in the solution. For any constant m , this is the case for a polynomial fraction of all keys, and can be ensure with subexponential overhead similar to

the procedure explained in Sect. 4.4. Further, we always use partial representations, i.e., the domains consist, similar to Sect. 4.2 and Sect. 4.3 of three parts of length $\frac{(1-\delta)n}{2}$, $\frac{(1-\delta)n}{2}$ and δn . Here we assume that each coordinate is present proportionally to the length of the segment, e.g., that the last segment contains each coordinate exactly $\frac{\delta n}{2m+1}$ times, which again can be ensured at the cost of a polynomial overhead only.

As outlined in Sect. 4.5, for each choice of m we now specify the used function domains and derive the amount representations of the solution. Let us start with the case of $m = 2$.

The Case of $m = 2$. We are looking for a solution $\mathbf{v} \in \{-2, \dots, 2\}$. For our first instantiation we use the same function definitions as in Sect. 4.3 given in Eqs. (8) and (9), where we choose a different α and β , specified later. Let us again specify the possible representations of each entry (similar to Eq. (10))

$$\begin{array}{l}
 0 : \quad \underbrace{0+0}_{z_0}, \quad \underbrace{1-1}_{z_1}, \quad \underbrace{-1+1}_{z_1}, \quad \underbrace{2-2}_{z_2}, \quad \underbrace{-2+2}_{z_2}, \\
 1 : \quad \underbrace{1+0}_{\frac{\delta n}{10}-o}, \quad \underbrace{0+1}_{\frac{\delta n}{10}-o}, \quad \underbrace{2-1}_o, \quad \underbrace{-1+2}_o, \\
 -1 : \quad \underbrace{-1+0}_{\frac{\delta n}{10}-o}, \quad \underbrace{0-1}_{\frac{\delta n}{10}-o}, \quad \underbrace{-2+1}_o, \quad \underbrace{1-2}_o, \\
 2 : \quad \underbrace{2+0}_{\frac{\delta n}{10}-\frac{t}{2}}, \quad \underbrace{0+2}_{\frac{\delta n}{10}-\frac{t}{2}}, \quad \underbrace{1+1}_t, \\
 -2 : \quad \underbrace{-2+0}_{\frac{\delta n}{10}-\frac{t}{2}}, \quad \underbrace{0-2}_{\frac{\delta n}{10}-\frac{t}{2}}, \quad \underbrace{-1-1}_t.
 \end{array}$$

Recall, that we have only representations on the last segment of length δn . As we expect any coordinate to be present $\delta n/5$ times, we need that the numbers below the representations in every row sum to $\delta n/5$. Therefore we have

$$z_0 + 2z_1 + 2z_2 = \delta n/5 \quad \Leftrightarrow \quad z_0 = \delta n/5 - 2z_1 - 2z_2.$$

Further by counting the respective number of ± 1 and ± 2 entries in those representations we obtain

$$\alpha = \frac{1}{10} + \frac{z_1 + t}{\delta} \quad \text{and} \quad \beta = \frac{1}{10} + \frac{z_2 - t/2 + o}{\delta},$$

while the number of representations is given as

$$R = \binom{\frac{\delta n}{5}}{z_0, z_1, z_1, z_2, z_2} \binom{\frac{\delta n}{5}}{\frac{\delta n}{10} - o, \frac{\delta n}{10} - o, o, o}^2 \binom{\frac{\delta n}{5}}{\frac{\delta n}{10} - \frac{t}{2}, \frac{\delta n}{10} - \frac{t}{2}, t}^2.$$

The values of z_1, z_2, o, t and δ are subject to numerical optimization.

Increased Representations for $m = 2$. In the following we represent \mathbf{v} on its last δn coordinates via the sum of two vectors $\mathbf{x}_0, \mathbf{x}_1 \in \{-3, \dots, 3\}^{\delta n}$. Similar

to including -2 and 2 entries in the case of $m = 1$ (Sect. 4.3), this leads to an increased amount of representations and in turn a runtime improvement.

First we naturally extend the definition $\mathcal{T}^n(\alpha, \beta)$ from Eq. (7) to $\mathcal{T}^n(\alpha, \beta, \gamma)$, where in the latter case included vectors contain exactly γn entries equal to ± 3 each. Then we let the new function domains be defined as

$$\begin{aligned} S_0 &:= \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times 0^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\delta n}(\alpha, \beta, \gamma) \quad \text{and} \\ S_1 &:= 0^{\frac{(1-\delta)n}{2}} \times \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha, \beta, \gamma), \end{aligned} \tag{11}$$

Accordingly we let their common image space be $S = \mathcal{T}^{\frac{(1-\delta)n}{2}}(1/3) \times \mathcal{T}^{\delta n}(\alpha, \beta, \gamma)$.

Now we obtain additional representations of any $0, \pm 1$ and ± 2 entry. Let us again specify all representations and how often they appear in the addition.

$$\begin{aligned} 0 : & \quad \underbrace{0+0}_{z_0}, & \underbrace{1-1}_{z_1}, & \underbrace{-1+1}_{z_1}, & \underbrace{2-2}_{z_2}, & \underbrace{-2+2}_{z_2}, & \underbrace{-3+3}_{z_3}, & \underbrace{-3+3}_{z_3}. \\ 1 : & \quad \underbrace{1+0}_{\frac{\delta n}{10}-o-d_1}, & \underbrace{0+1}_{\frac{\delta n}{10}-o-d_1}, & \underbrace{2-1}_o, & \underbrace{-1+2}_o, & \underbrace{3-2}_{d_1}, & \underbrace{-2+3}_{d_1}, \\ -1 : & \quad \underbrace{-1+0}_{\frac{\delta n}{10}-o-d_1}, & \underbrace{0-1}_{\frac{\delta n}{10}-o-d_1}, & \underbrace{-2+1}_o, & \underbrace{1-2}_o, & \underbrace{-3+2}_{d_1}, & \underbrace{2-3}_{d_1}, \\ 2 : & \quad \underbrace{2+0}_{\frac{\delta n}{10}-\frac{t}{2}-d_2}, & \underbrace{0+2}_{\frac{\delta n}{10}-\frac{t}{2}-d_2}, & \underbrace{1+1}_t, & \underbrace{3-1}_{d_2}, & \underbrace{-1+3}_{d_2}, \\ -2 : & \quad \underbrace{-2+0}_{\frac{\delta n}{10}-\frac{t}{2}-d_2}, & \underbrace{0-2}_{\frac{\delta n}{10}-\frac{t}{2}-d_2}, & \underbrace{-1-1}_t, & \underbrace{-3+1}_{d_2}, & \underbrace{1-3}_{d_2}. \end{aligned} \tag{12}$$

Analogously to before we have

$$z_0 + 2z_1 + 2z_2 + 2z_3 = \delta n/5 \quad \Leftrightarrow \quad z_0 = \delta n/5 - 2z_1 - 2z_2 - 2z_3.$$

Further by counting we obtain

$$\begin{aligned} \alpha &= \frac{1}{10} + \frac{z_1 + t - d_1 + d_2}{\delta}, & \beta &= \frac{1}{10} + \frac{z_2 - t/2 + o - d_2 + d_1}{\delta} \quad \text{and} \\ \gamma &= \frac{z_3 + d_1 + d_2}{\gamma} \end{aligned}$$

while the number of representations increases to

$$\begin{aligned} R &= \binom{\frac{\delta n}{5}}{z_0, z_1, z_1, z_2, z_2, z_3, z_3} \binom{\frac{\delta n}{5}}{\frac{\delta n}{10} - o - d_1, \frac{\delta n}{10} - o - d_1, o, o, d_1, d_1} \\ &\quad \cdot \binom{\frac{\delta n}{5}}{\frac{\delta n}{10} - \frac{t}{2} - d_2, \frac{\delta n}{10} - \frac{t}{2} - d_2, t, d_2, d_2}^2. \end{aligned}$$

The values of $z_1, z_2, z_3, o, t, d_1, d_2$ and δ are subject to numerical optimization.

Finally let us consider the case of $m = 3$.

The Case of $m = 3$. We now have a solution $\mathbf{v} \in \{-3, \dots, 3\}$. We represent this solution by using the same function domains as specified in Eq. (11), with an adapted choice of α, β and γ .

The possible representations stay therefore as specified in Eq. (12), by replacing $\frac{\gamma n}{10}$ by $\frac{\gamma n}{14}$. Since every row has now to add up to $\frac{\gamma n}{7}$ we obtain

$$z_0 + 2z_1 + 2z_2 + 2z_3 = \delta n/7 \iff z_0 = \delta n/7 - 2z_1 - 2z_2 - 2z_3.$$

We now get additionally representations for the ± 3 entries in \mathbf{v} :

$$\begin{aligned} 3 : & \quad \underbrace{3 + 0}_{\frac{\delta n}{14} - d_3}, \quad \underbrace{0 + 3}_{\frac{\delta n}{14} - d_3}, \quad \underbrace{2 + 1}_{d_3}, \quad \underbrace{1 + 2}_{d_3}, \\ -3 : & \quad \underbrace{-3 + 0}_{\frac{\delta n}{14} - d_3}, \quad \underbrace{0 - 3}_{\frac{\delta n}{14} - d_3}, \quad \underbrace{-2 - 1}_{d_3}, \quad \underbrace{-1 - 2}_{d_3}. \end{aligned}$$

This leads to the adapted choices of

$$\begin{aligned} \alpha &= \frac{1}{14} + \frac{z_1 + t - d_1 + d_2}{\delta}, \quad \beta = \frac{1}{14} + \frac{z_2 - t/2 + o - d_2 + d_1}{\delta} \quad \text{and} \\ \gamma &= \frac{1}{14} + \frac{z_3 + d_1 + d_2 - d_3}{\gamma}. \end{aligned}$$

Eventually the amount of representations is given as

$$\begin{aligned} R &= \binom{\frac{\delta n}{7}}{z_0, z_1, z_1, z_2, z_2, z_3, z_3} \binom{\frac{\delta n}{7}}{\frac{\delta n}{14} - o - d_1, \frac{\delta n}{14} - o - d_1, o, o, d_1, d_1}^2 \\ &\cdot \binom{\frac{\delta n}{7}}{\frac{\delta n}{14} - \frac{t}{2} - d_2, \frac{\delta n}{14} - \frac{t}{2} - d_2, t, d_2, d_2}^2 \binom{\frac{\delta n}{7}}{\frac{\delta n}{14} - d_3, \frac{\delta n}{14} - d_3, d_3, d_3}^2. \end{aligned}$$

References

1. Adj, G., Cervantes-Vázquez, D., Chi-Domínguez, J.J., Menezes, A., Rodríguez-Henríquez, F.: On the cost of computing isogenies between supersingular elliptic curves. In: Cid, C., Jacobson Jr., M.J. (eds.) SAC 2018. LNCS, vol. 11349, pp. 322–343. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-10970-7_15
2. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 411–439. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_14
3. Albrecht, M.R., et al.: Classic McEliece: conservative code-based cryptography (2020)

4. Banegas, G., et al.: CTIDH: faster constant-time CSIDH. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**(4), 351–387 (2021). <https://doi.org/10.46586/tches.v2021.i4.351-387>
5. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Krauthgamer, R. (ed.) 27th SODA, pp. 10–24. ACM-SIAM (Jan 2016). <https://doi.org/10.1137/1.9781611974331.ch2>
6. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{\frac{n}{20}}$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_31
7. Bellini, E., et al.: Parallel isogeny path finding with limited memory. In: INDOCRYPT 2022. LNCS, vol. 13774, pp. 294–316. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-22912-1_13
8. Bonnetain, X., Bricout, R., Schrottenloher, A., Shen, Y.: Improved classical and quantum algorithms for subset-sum. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 633–666. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_22
9. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 493–522. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_17
10. Bos, J., et al.: Crystals-kyber: a cca-secure module-lattice-based kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 353–367. IEEE (2018)
11. Both, L., May, A.: Decoding linear codes with high error rate and its impact for LPN security. In: Lange, T., Steinwandt, R. (eds.) PQCrypto 2018. LNCS, vol. 10786, pp. 25–46. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-79063-3_2
12. Bricout, R., Chailloux, A., Debris-Alazard, T., Lequesne, M.: Ternary syndrome decoding with large weight. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 437–466. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38471-5_18
13. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH (preliminary version). *IACR Cryptol. ePrint Arch*, p. 975 (2022). <https://eprint.iacr.org/2022/975>
14. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 395–427. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_15
15. Cervantes-Vázquez, D., Chenu, M., Chi-Domínguez, J.-J., De Feo, L., Rodríguez-Henríquez, F., Smith, B.: Stronger and faster side-channel protections for CSIDH. In: Schwabe, P., Thériault, N. (eds.) LATINCRYPT 2019. LNCS, vol. 11774, pp. 173–193. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30530-7_9
16. Chávez-Saab, J., Chi-Domínguez, J., Jaques, S., Rodríguez-Henríquez, F.: The SQALE of CSIDH: sublinear vélu quantum-resistant isogeny action with low exponents. *J. Cryptogr. Eng.* **12**(3), 349–368 (2022). <https://doi.org/10.1007/s13389-021-00271-w>
17. Chi-Domínguez, J., Rodríguez-Henríquez, F.: Optimal strategies for CSIDH. *Adv. Math. Commun.* **16**(2), 383–411 (2022). <https://doi.org/10.3934/amc.2020116>

18. Costello, C., Longa, P., Naehrig, M., Renes, J., Virdia, F.: Improved classical cryptanalysis of the computational supersingular isogeny problem. Cryptology ePrint Archive, Report 2019/298 (2019). <https://eprint.iacr.org/2019/298>
19. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006). <https://eprint.iacr.org/2006/291>
20. Esser, A.: Revisiting nearest-neighbor-based information set decoding. Cryptology ePrint Archive, Report 2022/1328 (2022). <https://eprint.iacr.org/2022/1328>
21. Esser, A., Girme, R., Mukherjee, A., Sarkar, S.: Memory-efficient attacks on small lwe keys. Cryptology ePrint Archive (2023)
22. Esser, A., May, A.: Low weight discrete logarithm and subset sum in $2^{0.65n}$ with polynomial memory. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 94–122. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_4
23. Esser, A., May, A., Zweyding, F.: McEliece needs a break - solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 433–457. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-07082-2_16
24. Galbraith, S.D., Hess, F., Smart, N.P.: Extending the GHS weil descent attack. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 29–44. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_3
25. Glaser, T., May, A.: How to enumerate LWE keys as narrow as in kyber/dilithium. Cryptology ePrint Archive, Report 2022/1337 (2022). <https://eprint.iacr.org/2022/1337>
26. Hutchinson, A., LeGrow, J., Koziel, B., Azarderakhsh, R.: Further optimizations of CSIDH: a systematic approach to efficient strategies, permutations, and bound vectors. In: Conti, M., Zhou, J., Casalichio, E., Spognardi, A. (eds.) ACNS 2020. LNCS, vol. 12146, pp. 481–501. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57808-4_24
27. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
28. Kuperberg, G.: A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. SIAM J. Comput. **35**(1), 170–188 (2005)
29. Maino, L., Martindale, C.: An attack on SIDH with arbitrary starting curve. IACR Cryptol. ePrint Arch., p. 1026 (2022). <https://eprint.iacr.org/2022/1026>
30. May, A.: How to meet ternary LWE keys. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 701–731. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_24
31. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_6
32. May, A., Ozerov, I.: A generic algorithm for small weight discrete logarithms in composite groups. In: Joux, A., Youssef, A. (eds.) SAC 2014. LNCS, vol. 8781, pp. 278–289. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13051-4_17
33. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 203–228. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_9

34. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. The deep space network progress report 42–44, Jet Propulsion Laboratory, California Institute of Technology (Jan/Feb 1978). https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF
35. Meyer, M., Campos, F., Reith, S.: On lions and elligators: an efficient constant-time implementation of CSIDH. In: Ding, J., Steinwandt, R. (eds.) PQCrypto 2019. LNCS, vol. 11505, pp. 307–325. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25510-7_17
36. Onuki, H., Aikawa, Y., Yamazaki, T., Takagi, T.: (Short Paper) a faster constant-time algorithm of CSIDH keeping two points. In: Attrapadung, N., Yagi, T. (eds.) IWSEC 2019. LNCS, vol. 11689, pp. 23–33. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26834-3_2
37. Onuki, H., Aikawa, Y., Yamazaki, T., Takagi, T.: A constant-time algorithm of CSIDH keeping two points. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **103-A**(10), 1174–1182 (2020). <https://doi.org/10.1587/transfun.2019DMP0008>
38. Peikert, C.: He gives C-sieves on the CSIDH. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 463–492. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_16
39. Prange, E.: The use of information sets in decoding cyclic codes. IRE Trans. Inf. Theory **8**(5), 5–9 (1962)
40. Robert, D.: Breaking SIDH in polynomial time. IACR Cryptol. ePrint Arch. p. 1038 (2022). <https://eprint.iacr.org/2022/1038>
41. Rostovtsev, A., Stolbunov, A.: Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145 (2006). <https://eprint.iacr.org/2006/145>
42. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: 35th FOCS, pp. 124–134. IEEE Computer Society Press (Nov 1994). <https://doi.org/10.1109/SFCS.1994.365700>
43. Tani, S.: Claw finding algorithms using quantum walk. Theoret. Comput. Sci. **410**(50), 5285–5297 (2009)
44. van Hoof, I., Kirshanova, E., May, A.: Quantum key search for ternary LWE. In: Cheon, J.H., Tillich, J.-P. (eds.) PQCrypto 2021 2021. LNCS, vol. 12841, pp. 117–132. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81293-5_7
45. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. J. Cryptol. **12**(1), 1–28 (1999). <https://doi.org/10.1007/PL00003816>

Encryption



On the Complete Non-malleability of the Fujisaki-Okamoto Transform

Daniele Friolo¹(✉), Matteo Salvino²(✉), and Daniele Venturi¹(✉)

¹ Department of Computer Science, Sapienza University of Rome, Rome, Italy
{friolo,venturi}@di.uniroma1.it

² Research Institute CODE, Universität der Bundeswehr München,
Munich, Germany
matteo.salvino@unibw.de

Abstract. The Fujisaki-Okamoto (FO) transform (CRYPTO 1999 and JoC 2013) turns any weakly (i.e., IND-CPA) secure public-key encryption (PKE) scheme into a strongly (i.e., IND-CCA) secure key encapsulation method (KEM) in the random oracle model (ROM). Recently, the FO transform re-gained momentum as part of CRISTAL-Kyber, selected by the NIST as the PKE winner of the post-quantum cryptography standardization project.

Following Fischlin (ICALP 2005), we study the *complete non-malleability* of KEMs obtained via the FO transform. Intuitively, a KEM is completely non-malleable if no adversary can maul a given public key and ciphertext into a new public key and ciphertext encapsulating a related key for the underlying blockcipher.

On the negative side, we find that KEMs derived via FO are *not* completely non-malleable in general. On the positive side, we show that complete non-malleability holds in the ROM by assuming the underlying PKE scheme meets an additional property, or by a slight tweak of the transformation.

Keywords: Non-malleability · Key encapsulation · Public-key cryptography

1 Introduction

Public-key encryption (PKE) allows Alice to encrypt a message under a Bob's public key, so that Bob can decrypt the ciphertext using the corresponding secret key. Several security notions for PKE have been proposed in the literature. The most basic one, namely *indistinguishability against chosen-plaintext attacks* (IND-CPA) requires that an adversary, given the public key, cannot distinguish between the encryption of two messages.

M. Salvino—The work was carried out whilst the author was a student at Sapienza University of Rome, Rome, IT.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
M. Tibouchi and X. Wang (Eds.): ACNS 2023, LNCS 13906, pp. 307–335, 2023.
https://doi.org/10.1007/978-3-031-33491-7_12

Non-malleability. As noted for the first time by Dolev *et al.* [9], IND-CPA appears to be insufficient for many applications. Consider for instance the setting of private auctions. Here, a bidder can sample its own pair of public/secret keys, encrypt the bid b using the public key, and send the encryption together with the public key to the auctioneer. After all the participants have sent their bid, the auctioneer can declare the winner by asking each party to reveal the secret key (or the bid itself, along with the random coins used for encryption). A malicious user, given a ciphertext c containing the bid of another party, can try to construct a ciphertext c' that, when decrypted, leads to a bid b' such that $b' > b$.

In light of such malleability attacks, stronger security notions for PKE schemes have been introduced. These include the notions of *non-malleability under chosen-plaintext* and *chosen-ciphertext attacks* [3, 6, 9, 19] (NM-CPA and NM-CCA), and *indistinguishability under chosen-ciphertext attacks* (IND-CCA). All of these notions imply that the attacker, given the public key and a target ciphertext, is unable to craft a mangled ciphertext whose underlying plaintext is related to the one contained in the target ciphertext.

Complete Non-malleability. In 2005, Fischlin [12] noted that non-malleability might be still insufficient for some applications. In fact, the above notions do not account for the possibility that the attacker may try to maul the public key as well. For instance, consider again the setting of private auctions. A malicious user, knowing a ciphertext c and the public key pk , may try to craft a public key pk' and a ciphertext c' which encrypt a bid $b' > b$. To capture these attacks, Fischlin introduced *complete non-malleability*, which rules out such adversaries.

As noted by Fischlin himself, completely non-malleable PKE has several useful applications, including key-agreement protocols with security against unknown key attacks, and signature schemes with security against strong unforgeability attacks (as needed, e.g., in e-cash systems).

Known Constructions. Fischlin [12] showed that a simple variant of RSA-OAEP is completely non-malleable in the random oracle model (ROM).¹ Ventre and Visconti [23] later gave two constructions of completely non-malleable PKE in the common reference string (CRS) model, based on non-interactive zero-knowledge (NIZK) proofs for all of NP.

Subsequent work provided more efficient constructions of completely non-malleable PKE without random oracles, using both pairing-based assumptions [7, 16] and lattice-based assumptions [21, 22].

1.1 Our Contributions

In practice, due to its computational overhead, PKE is never used to encrypt long messages. Rather, as it happens in many real-world protocols (including TLS), the parties use public-key techniques in order to establish a common

¹ He also proves that the original version of RSA-OAEP, as well as the Cramer-Shoup PKE [8], is not completely non-malleable.

secret key for a blockcipher, which can be later used in order to encrypt any subsequent communication of arbitrary length. In the literature this paradigm is also known as the *key/data encapsulation method* (KEM/DEM), or simply hybrid encryption. This motivates our main question:

Can we get efficient constructions of completely non-malleable PKE via the KEM/DEM paradigm?

Our main contribution is a positive answer to the above question. Namely, we put forward natural notions of complete non-malleability for KEMs and show that these notions are sufficient to imply completely non-malleable PKE with small ciphertext rate. Furthermore, we show that an already existing, widely-used, KEM meets our notions. We elaborate on these contributions below.

Definitions. A *key encapsulation method* (KEM) is made of two algorithms: An encapsulation algorithm that, given the public key \mathbf{pk} , outputs a ciphertext c encapsulating a secret key K ; and a decapsulation algorithm that, given the secret key \mathbf{sk} corresponding to \mathbf{pk} , allows to recover K . Similarly to PKE, several non-malleability properties for KEMs have been introduced.

In Sect. 3, we put forward three indistinguishability-based variants of completely non-malleable KEMs (dubbed NM-CPA*, NM-CCA1* and NM-CCA2*), capturing different flavors of chosen-plaintext and chosen-ciphertext attacks.

In the full version [13], we further define the corresponding simulation-based variants (dubbed SNM-CPA*, SNM-CCA1* and SNM-CCA2*), and show the equivalence between the NM-ATK* and SNM-ATK* notions for $\text{ATK} \in \{\text{CPA}, \text{CCA1}, \text{CCA2}\}$. More specifically, we show that for NM-CPA* and SNM-CPA* the equivalence holds for so-called *complete relations*, while for NM-CCA1* and SNM-CCA1*, and for NM-CCA2* and SNM-CCA2*, the equivalence holds for a restricted set of relations called *lacking relations*. These findings are in line with the work by Ventre and Visconti [23], who showed analogous results for completely non-malleable PKE.

Analysis of Fujisaki-Okamoto. As our main contribution, we analyze the complete non-malleability of the Fujisaki-Okamoto (FO) transform [14]. Recall that the FO transform turns any IND-CPA secure PKE into an IND-CCA secure KEM in the ROM, without affecting the ciphertext size, and at the cost of a very small extra computation effort w.r.t. the underlying PKE scheme (when the RO is replaced with a real-world hash function like SHA-256). Recently, the FO transform re-gained momentum as part of CRISTAL-Kyber, selected by the NIST as the PKE winner of the post-quantum cryptography standardization project [18]. In this light, we believe that investigating further security properties of the FO transform is a very natural research question.

Our analysis follows the modular analysis of the FO transform due to Hofheinz *et al.* [15]. Here, one interprets the FO transform as a sequence of two transformations \mathbb{T} and \mathbb{U} :

- The transformation T starts with any IND-CPA PKE. The encryption algorithm runs the encryption algorithm of the underlying PKE scheme but sets its randomness to $G(m)$, where G is a RO. The decryption algorithm runs the decryption algorithm of the underlying PKE scheme, and returns \perp if the decrypted message m' is \perp or if the encryption of m' with randomness $G(m')$ does not equal the ciphertext.
- The transformation U takes a PKE scheme satisfying different flavours of one-wayness (which are achieved by the transformation T), and outputs an IND-CCA secure KEM. This transformation essentially comes in 2 variants.² U_m calculates the encapsulated key K by randomly choosing a message m from the message space of the underlying PKE scheme, encrypting m under pk , and then computing K as $H(m)$, where H is a RO. U instead computes the key as $H(m, c)$.

First, in Sect. 4.1, we show a concrete attack against the transformation U_m that works even when considering the weakest flavour of complete non-malleability (i.e., NM-CPA*). We take the El-Gamal PKE scheme as the base PKE scheme to be transformed by T and U into a KEM scheme. In particular, we prove that an adversary can appropriately maul the public key pk and the ciphertext c encrypting m , and come up with a ciphertext c' encrypting the same message m under a different public key pk' . Since the encapsulated key computed by U_m is $H(m)$, the key encapsulated by c and c' will be the same. Thus, complete non-malleability is trivially broken.

Second, in Sect. 4.2, we show that the transformation U is not completely non-malleable. To see this, it suffices to take a contrived PKE scheme in which we add a dummy bit to the public key of a PKE scheme satisfying the one-wayness properties required by the transformation U . This additional bit is completely ignored by the encryption algorithm and does not effect one-wayness. However, one can trivially break complete non-malleability by flipping the last bit of the public key. On the positive side, we show that U does achieve complete non-malleability assuming the underlying PKE scheme meets a natural public-key uniqueness property, where the latter essentially means that an adversary cannot come up with different public keys pk, pk' for which there exist a message m and a ciphertext c such that c is a valid encryption of m under both pk and pk' . Indeed, we point out that uniqueness seems to be a standard property to achieve non-malleability, e.g. quasi-unique responses for Fiat-Shamir signatures [11, 12].

Finally, in Sect. 4.3, we show how to tweak the transform U in order to obtain complete non-malleability without requiring public-key uniqueness. For this, it suffices to compute the key K as $H(m, c, pk)$. This way, even if the attacker can break public-key uniqueness, the random oracle will ensure that the two encapsulated keys are independent. We notice that a similar technique was already used in [10], where regular CCA security of the FO transform in the multi-user setting is achieved by adding just a fraction of the public key with high

² Each of U and U_m also comes in 2 variants, but the difference is irrelevant for what follows.

min-entropy as an input to the hash function. However, it is not clear whether complete non-malleability of FO is achievable with such a slight modification.

Relation with Completely Non-malleable PKE. In Sect. 5, we show that by combining a completely non-malleable KEM with a non-malleable secret-key encryption (SKE) scheme we obtain a completely non-malleable PKE using the KEM/DEM paradigm. Furthermore, we observe that one can always obtain a completely non-malleable KEM by encrypting a random secret key via a completely non-malleable PKE.

1.2 Related Work

Nagao *et al.* in [17] analyze standard non-malleability in the context of key encapsulation. In particular, they consider different flavours of non-malleable KEM, such as NM-CPA, NM-CCA1 and NM-CCA2.

Ventre and Visconti [23] note that the stronger CCA2 notion of complete non-malleability is not strictly necessary for some of the applications proposed by Fischlin [12]. Hence, they put forward weaker flavours of complete non-malleability, and establish the relations between the standard comparison-based notions NM-ATK* and their simulation-based counterparts SNM-ATK* for $\text{ATK} \in \{\text{CPA}, \text{CCA1}, \text{CCA2}\}$. They also give two constructions of completely non-malleable NM-CCA2* secure PKE: one in the CRS model using NIZK proofs for all of NP, and one in the plain model using interactive encryption.

Barbosa and Farshim [1] consider an equivalent indistinguishability-based notion of complete non-malleability based on so-called strong CCA security, in which the (strong) decryption oracle provides decryptions under arbitrarily chosen public keys. Duman *et al.* [10] analyze CCA security of the FO transform in the multi-user setting.

2 Preliminaries

In this section we introduce some basic notation and recall a few standard definitions that will be used later to prove some of our results.

2.1 Notation

We use calligraphic letters to denote sets, such as \mathcal{X} , and lower-case letters for variables, such as x . We use $x \leftarrow_s \mathcal{X}$ to indicate that x is picked uniformly at random from \mathcal{X} . A similar notation is used in the presence of a randomized or probabilistic algorithm A . Indeed, $x \leftarrow_s A(\cdot)$ means that x is the output of the randomized algorithm A . Alternatively, when a random coin r is given as an input of A , we equivalently write $x := A(\cdot; r)$. All the algorithms we will consider are PPT (Probabilistic Polynomial Time), i.e. for any input $x \in \{0, 1\}^*$ and random coin r , $A(x; r)$ terminates in at most polynomial many steps, in the size of its inputs. When an algorithm A has access to a set of oracles $\{O_1, \dots, O_n\}$, we use

the notation A^{O_1, \dots, O_n} , to indicate that A can interact in a black-box manner with oracles O_1, \dots, O_n during its computation. We denote with $\lambda \in \mathbb{N}$ the security parameter and we will assume that all the algorithms we will consider take λ as an input. A function $\nu : \mathbb{N} \rightarrow [0, 1]$ is negligible if for every polynomial $p(n) \exists N \in \mathbb{N}$ s.t. $\forall n_0 \geq N, \nu(n_0) < \frac{1}{p(n_0)}$. We denote with $\text{negl}(\lambda)$ any function that is negligible in λ . Given two random variables X and Y , we denote $X \approx_c Y$ when X and Y are computationally indistinguishable, and with $X \equiv Y$ when X and Y are identically distributed.

2.2 Public-Key Encryption

A public-key encryption (PKE) scheme Π consists of three algorithms ($\text{Gen}, \text{Enc}, \text{Dec}$), together with a message space \mathcal{M} (which we assume to be efficiently recognizable) where:

- The key generation algorithm Gen takes as input 1^λ and outputs a public-private key pair (pk, sk) .
- The encryption algorithm Enc takes as inputs a public key pk and a message $m \in \mathcal{M}$, and outputs an encryption c of the message m under pk .
- The deterministic decryption algorithm Dec takes as inputs a decryption key sk and a ciphertext c , and outputs either a message $m \in \mathcal{M}$, or \perp (denoting failure).

Next, we define both correctness and security of PKE as needed for our purposes. Some of the definitions below are taken verbatim from [15].

Definition 1 (γ -uniformity). *Let $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ be a PKE scheme with message space \mathcal{M} . Given $(\text{pk}, \text{sk}) \leftarrow_s \text{Gen}(1^\lambda)$, a message $m \in \mathcal{M}$ and a ciphertext c we define the γ -uniformity function as follows*

$$\gamma(m, c) = \Pr [c = \text{Enc}(\text{pk}, m)],$$

where the probability is taken over the choice of the random coins used for encrypting m under pk .

We say that Π is γ -uniform if, for any $(\text{pk}, \text{sk}) \in \text{Gen}(1^\lambda)$, any message $m \in \mathcal{M}$, and any ciphertext $c \in \{0, 1\}^*$, it holds that $\gamma(m, c) \leq \gamma$.

Definition 2 (δ -correctness). *Let $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ be a PKE scheme with message space \mathcal{M} . A PKE scheme $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ is δ -correct if*

$$\mathbf{E} \left[\max_{m \in \mathcal{M}} \Pr [\text{Dec}(\text{sk}, c) \neq m \mid c \leftarrow_s \text{Enc}(\text{pk}, m)] \right] \leq \delta,$$

where the expectation is taken over the sampling of $(\text{pk}, \text{sk}) \leftarrow_s \text{Gen}(1^\lambda)$.

OW-PCA and OW-PCVA. We recall the definitions of one-wayness under plaintext checking attacks (OW-PCA) and one-wayness under plaintext and validity checking attacks (OW-PCVA).

Definition 3 (OW-ATK). Let $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ be a public-key encryption scheme with message space \mathcal{M} . We define the following $\text{PKE}^{\text{ow-atk}}$ games for $\text{atk} \in \{\text{pca}, \text{pcva}\}$

<p>Experiment $\text{PKE}_{\Pi, \mathcal{A}}^{\text{ow-atk}}(\lambda)$</p> <hr style="border: 0; border-top: 1px solid black; margin: 5px 0;"/> <p>$(\text{pk}^*, \text{sk}^*) \leftarrow_{\\$} \text{Gen}(1^\lambda)$ $m^* \leftarrow_{\\$} \mathcal{M}$ $c^* \leftarrow_{\\$} \text{Enc}(\text{pk}^*, m^*)$ $m' \leftarrow_{\\$} \mathcal{A}^{\text{O}_1}(\text{pk}, c^*)$ return $\text{PCO}(\text{sk}^*, m', c^*)$</p>	<p>Oracle $\text{PCO}(\text{sk}^*, m, c)$</p> <hr style="border: 0; border-top: 1px solid black; margin: 5px 0;"/> <p>return 1 iff $(\text{Dec}(\text{sk}^*, c) = m) \wedge (m \neq \perp)$</p> <p>Oracle $\text{CVO}^{(c^*)}(\text{sk}^*, c)$</p> <hr style="border: 0; border-top: 1px solid black; margin: 5px 0;"/> <p>$m := \text{Dec}(\text{sk}^*, c)$ return 1 iff $m \neq \perp$</p>
--	---

In the experiment above,

- if $\text{atk} = \text{pca}$ then $\text{O}_1 = \text{PCO}(\text{sk}^*, \cdot, \cdot)$,
- if $\text{atk} = \text{pcva}$ then $\text{O}_1 = \text{PCO}(\text{sk}^*, \cdot, \cdot), \text{CVO}^{(c^*)}(\text{sk}^*; \cdot)$,

where $\text{CVO}^{(c^*)}(\text{sk}, \cdot)$ means that \mathcal{A} is allowed to query the CVO algorithm for any ciphertext distinct from the challenge ciphertext c^* . We say that Π is OW-ATK secure if for all PPT \mathcal{A} , $\Pr [\text{PKE}_{\Pi, \mathcal{A}}^{\text{ow-atk}}(\lambda) = 1] \leq \text{negl}(\lambda)$.

Complete Non-malleability. Finally, we recall the indistinguishability-based security definition for completely non-malleable PKE as defined by Ventre and Visconti [23].

Definition 4 (NM-CPA*, NM-CCA1*, NM-CCA2*). Let $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ be a public-key encryption scheme, and let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT adversary. For $\text{atk} \in \{\text{cpa}, \text{cca1}, \text{cca2}\}$ and $\lambda \in \mathbb{N}$ then we have that $\text{PKE}_{\Pi, \mathcal{A}}^{\text{nm-atk*}}(\lambda) \approx_c \text{PKE}_{\Pi, \mathcal{A}, \$}^{\text{nm-atk*}}(\lambda)$. The experiments $\text{PKE}_{\Pi, \mathcal{A}}^{\text{nm-atk*}}(\lambda)$ and $\text{PKE}_{\Pi, \mathcal{A}, \$}^{\text{nm-atk*}}(\lambda)$ are defined as follows:

Experiment $\text{PKE}_{\Pi, \mathbf{A}}^{nm-atk^*}(\lambda)$

$(\text{pk}^*, \text{sk}^*) \leftarrow \text{Gen}(1^\lambda)$
 $(\mathcal{M}, s) \leftarrow \text{A}_1^{\text{O}1}(\text{pk})$
 $m^* \leftarrow \mathcal{M}$
 $c^* \leftarrow \text{Enc}(\text{pk}^*, m^*)$
 $(\text{pk}, R, c) \leftarrow \text{A}_2^{\text{O}2}(\mathcal{M}, \text{pk}^*, s, c^*)$
return 1 iff $\exists(m, r)$ s.t.
 $(c = \text{Enc}(\text{pk}, m; r)) \wedge$
 $(c \neq c^* \vee \text{pk} \neq \text{pk}^*) \wedge$
 $(m \neq \perp) \wedge R(m, m^*, \text{pk}, \text{pk}^*, c)$

Experiment $\text{PKE}_{\Pi, \mathbf{A}, \$}^{nm-atk^*}(\lambda)$

$(\text{pk}^*, \text{sk}^*) \leftarrow \text{Gen}(1^\lambda)$
 $(\mathcal{M}, s) \leftarrow \text{A}_1^{\text{O}1}(\text{pk})$
 $m^*, \tilde{m} \leftarrow \mathcal{M}$
 $c^* \leftarrow \text{Enc}(\text{pk}^*, m^*)$
 $(\text{pk}, R, c) \leftarrow \text{A}_2^{\text{O}2}(\mathcal{M}, \text{pk}^*, s, c^*)$
return 1 iff $\exists(m, r)$ s.t.
 $(c = \text{Enc}(\text{pk}, m; r)) \wedge$
 $(c \neq c^* \vee \text{pk} \neq \text{pk}^*) \wedge$
 $(m \neq \perp) \wedge R(m, \tilde{m}, \text{pk}, \text{pk}^*, c)$

In the experiments above

if $atk = cpa$ then $\text{O}_1 = \epsilon$ and $\text{O}_2 = \epsilon$,
 if $atk = cca1$ then $\text{O}_1 = \text{Dec}(\text{sk}^*, \cdot)$ and $\text{O}_2 = \epsilon$,
 if $atk = cca2$ then $\text{O}_1 = \text{Dec}(\text{sk}^*, \cdot)$ and $\text{O}_2 = \text{Dec}^{(c^*)}(\text{sk}^*, \cdot)$,

where $\text{Dec}^{(c)}(\text{sk}, \cdot)$ means that \mathbf{A} is allowed to query Dec oracle for any ciphertext distinct from the challenge ciphertext c^* .

2.3 Secret-Key Encryption

A secret-key encryption scheme (SKE) consists of a triple of algorithm $(\text{Gen}, \text{Enc}, \text{Dec})$ together with a message space \mathcal{M} and a key space \mathcal{K} , in which:

- The key generation algorithm Gen takes as input 1^λ and outputs a secret key $\text{K} \in \mathcal{K}$.
- The encryption algorithm Enc takes as input the secret key $\text{K} \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext c .
- The decryption algorithm Dec takes as input the secret key $\text{K} \in \mathcal{K}$ and a ciphertext c , and outputs a message $m \in \mathcal{M}$, or \perp denoting failure.

Let us consider the flavour of correctness needed for our scopes.

Definition 5 (Correctness). A SKE scheme $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ is correct if

$$\Pr [\text{Dec}(\text{sk}, c) = m \mid \text{K} \leftarrow \text{Gen}(1^\lambda); c \leftarrow \text{Enc}(\text{K}, m)] = 1.$$

To make our security proofs go through, we use the non-malleability security definition NM-ATK with $\text{ATK} \in \{\text{CPA}, \text{CCA1}, \text{CCA2}\}$ introduced by Bellare et al. [2]. As highlighted in [4, 5] the original definitions were introduced the asymmetric setting [6, 9, 20] but can be “lifted” to the symmetric setting using the encryption oracle based template of [2]. Hence, by leveraging the results of Bellare et al [6], NM-ATK for $\text{ATK} \in \{\text{CPA}, \text{CCA1}, \text{CCA2}\}$ is equivalent to the indistinguishability-based counterpart IND-ATK [4–6].

Definition 6 (NM-CPA, NM-CCA1, NM-CCA2). *Given a set of relations \mathcal{R} , a SKE scheme Π is NM-ATK secure with respect to any relation $R \in \mathcal{R}$, if for any NM-ATK adversary $A = (A_0, A_1)$, $\text{SKE}_{\Pi, A}^{nm-atk}(\lambda) \approx_c \text{SKE}_{\Pi, A, \$}^{nm-atk}(\lambda)$. The experiments are defined as follows*

<i>Experiment</i> $\text{SKE}_{\Pi, A}^{nm-atk}(\lambda)$	<i>Experiment</i> $\text{SKE}_{\Pi, A, \$}^{nm-atk}(\lambda)$
$K^* \leftarrow \$ \mathcal{K}$	$K^* \leftarrow \$ \mathcal{K}$
$(\mathcal{M}, s) \leftarrow \$ A_0^{O_1}(1^\lambda)$	$(\mathcal{M}, s) \leftarrow \$ A_0^{O_1}(1^\lambda)$
$m^* \leftarrow \$ \mathcal{M}$	$m^*, \tilde{m} \leftarrow \$ \mathcal{M}$
$c^* \leftarrow \$ \text{Enc}(K^*, m^*)$	$c^* \leftarrow \$ \text{Enc}(K^*, m^*)$
$(R, c') \leftarrow \$ A_1^{O_2}(\mathcal{M}, s, c^*)$	$(R, c') \leftarrow \$ A_1^{O_2}(\mathcal{M}, s, c^*)$
$m' := \text{Dec}(K^*, c')$	$m' := \text{Dec}(K^*, c')$
return 1 <i>iff</i>	return 1 <i>iff</i>
$(m' \neq \perp) \wedge (c' \neq c^*) \wedge R(m^*, m')$	$(m' \neq \perp) \wedge (c' \neq c^*) \wedge R(\tilde{m}, m')$

In the experiments above

- if* $atk = cpa$ *then* $O_1 = \text{Enc}(K^*, \cdot)$ *and* $O_2 = \text{Enc}(K^*, \cdot)$,
- if* $atk = cca1$ *then* $O_1 = \text{Enc}(K^*, \cdot), \text{Dec}(K^*, \cdot)$ *and* $O_2 = \text{Enc}(K^*, \cdot)$,
- if* $atk = cca2$ *then* $O_1 = \text{Enc}(K^*, \cdot), \text{Dec}(K^*, \cdot)$ *and* $O_2 = \text{Enc}(K^*, \cdot), \text{Dec}^{(c^*)}(K^*, \cdot)$,

where $\text{Dec}^{(c^*)}(K^*, \cdot)$ means that A is allowed to query the Dec oracle for any ciphertext c distinct from the challenge ciphertext c^* .

3 Completely Non-malleable KEMs

In this section, we formally introduce key encapsulation methods (KEMs). Then, following the work on completely non-malleable PKE schemes of Fischlin [12], and then Ventre and Visconti [23], we extend the notions of complete non-malleability of PKEs to the context of KEMs. We follow a similar blueprint of [23] (that, in turn, bases its definitions on [3]) by introducing three indistinguishability-based security notions for completely non-malleable KEMs dubbed NM-CPA*, NM-CCA1* and NM-CCA2*. We further introduce, in the full version [13], three simulation-based notions dubbed SNM-CPA*, SNM-CCA1* and SNM-CCA2* and investigate the relationship between indistinguishability-based and the simulation-based notions.

A KEM scheme consists of a triple of algorithms $\Pi = (\text{Gen}, \text{Encaps}, \text{Decaps})$, together with a key space \mathcal{K} , in which:

- The key generation algorithm Gen takes as input 1^λ and outputs a public-private key pair (pk, sk) .

- The encapsulation algorithm **Encaps** takes as input a public key pk , and outputs a ciphertext c as well as a key K .
- The decapsulation algorithm **Decaps** takes as input a private key sk and a ciphertext c and returns a key $K \in \mathcal{K}$ or \perp (denoting failure).

First of all, we start by considering the flavour of correctness needed for our scopes.

Definition 7 (ϵ -correctness). *A KEM scheme $\Pi = (\text{Gen}, \text{Encaps}, \text{Decaps})$ is ϵ -correct if*

$$\Pr [\text{Decaps}(\text{sk}, c) \neq K \mid (\text{pk}, \text{sk}) \leftarrow_s \text{Gen}(1^\lambda); (c, K) \leftarrow_s \text{Encaps}(\text{pk})] \leq \epsilon.$$

Before diving into the definitions complete non-malleability for KEMs, we introduce the notion of complete relation for KEMs, firstly defined by Fischlin [12] in the setting of completely non-malleable PKE. A complete relation R in the KEM setting is a probabilistic algorithm taking as inputs two public keys pk and pk' , two encapsulation keys K and K^* , and a ciphertext c' . It outputs 1 if the relation is satisfied, and 0 otherwise. We will refer as \mathcal{R} to be the set of complete relations.

In the indistinguishability-based notion of complete non-malleability, we ask the adversary to distinguish between two experiments. In both of them, we let the adversary learn the challenge public key, ciphertext and encapsulated key, and then make the adversary output a new public key pk' , a relation R and a new ciphertext c' . If there exists a key $K' \neq K$ and randomness r such that c' and K' can be obtained by running the encapsulation algorithm with $\text{pk}' \neq \text{pk}^*$ and randomness r , then the experiment will output 1. In the left-side experiment the adversary will receive the key K^* encapsulated in c^* , while in the right-side experiment the key K^* is sampled randomly and hence totally unrelated from the key encapsulated in the received ciphertext. Note that the adversary may come up with a triple (pk', R, c') such that there exists a key encapsulated in c' satisfying the relation $R(K', K^*, \text{pk}', \text{pk}^*, c')$, but the adversary may have negligible advantage in distinguishing whether the key was encapsulated in c^* or it was randomly chosen.

Definition 8 (NM-CPA*, NM-CCA1*, NM-CCA2*). *Given a set of relations \mathcal{R} , a key-encapsulation mechanism $\Pi = (\text{Gen}, \text{Encaps}, \text{Decaps})$ is NM-ATK* secure with respect to any relation $R \in \mathcal{R}$, if for any NM-ATK* adversary $A = (A_1, A_2)$ for $\text{atk} \in \{\text{cpa}, \text{cca1}, \text{cca2}\}$ and for all $\lambda \in \mathbb{N}$, $\text{KEM}_{\Pi, A}^{\text{nm-atk}^*}(\lambda) \approx_c \text{KEM}_{\Pi, A, \mathcal{R}}^{\text{nm-atk}^*}(\lambda)$.*

The experiments are defined as follows:

<p>Experiment $\text{KEM}_{II,A}^{nm-atk^*}(\lambda)$</p> <hr/> <p>$(pk^*, sk^*) \leftarrow_s \text{Gen}(1^\lambda)$ $st \leftarrow_s A_1^{O_1}(pk^*)$ $(c^*, K^*) \leftarrow_s \text{Encaps}(pk^*)$</p> <p>$(pk', R, c') \leftarrow_s A_2^{O_2}(pk^*, c^*, K^*, st)$ return 1 if $\exists(K', r)$ such that $((c', K') = \text{Encaps}(pk'; r)) \wedge$ $(pk' \neq pk^* \vee K' \neq K^*) \wedge (K' \neq \perp)$ $\wedge R(K', K^*, pk', pk^*, c')$</p>	<p>Experiment $\text{KEM}_{II,A,S}^{nm-atk^*}(\lambda)$</p> <hr/> <p>$(pk, sk) \leftarrow_s \text{Gen}(1^\lambda)$ $st \leftarrow_s A_1^{O_1}(pk^*)$ $K^* \leftarrow_s \{0, 1\}^\lambda$ $(\hat{c}, \hat{K}) \leftarrow_s \text{Encaps}(pk^*)$ $(pk', R, c') \leftarrow_s A_2^{O_2}(pk^*, \hat{c}, K^*, st)$ return 1 if $\exists(K', r)$ such that $((c', K') = \text{Encaps}(pk'; r)) \wedge$ $(pk' \neq pk^* \vee K' \neq K^*) \wedge (K' \neq \perp)$ $\wedge R(K', K^*, pk', pk^*, c')$</p>
--	--

In the experiments above,

$$\begin{aligned}
 &\text{if } atk = cpa \text{ then } O_1 = \epsilon && \text{and } O_2 = \epsilon, \\
 &\text{if } atk = cca1 \text{ then } O_1 = \text{Decaps}(sk^*, \cdot) \text{ and } O_2 = \epsilon, \\
 &\text{if } atk = cca2 \text{ then } O_1 = \text{Decaps}(sk^*, \cdot) \text{ and } O_2 = \text{Decaps}^{(c^*)}(sk^*, \cdot),
 \end{aligned}$$

where $\text{Decaps}^{(c^*)}(sk, \cdot)$ means that A is allowed to query Decaps algorithm for any ciphertext distinct from the challenge ciphertext c^* .

In [12], Fischlin showed that there exist encryption and signatures schemes that are not NM-CCA2* secure, even though they are NM-CCA2 secure, i.e. the CCA secure in standard non-malleability notion of PKE (see [6] for the formal definition). As we show in the following theorem, this holds also in the case of KEMs. Let $\text{ATK} \in \{\text{CPA}, \text{CCA1}, \text{CCA2}\}$.

Theorem 1. *Assume that there exists a NM-ATK secure KEM $\Pi = (\text{Gen}, \text{Encaps}, \text{Decaps})$; then there exists a NM-ATK secure KEM $\Pi' = (\text{Gen}', \text{Encaps}', \text{Decaps}')$ which is not NM-ATK* secure.*

The proof of the theorem appears in Appendix A.1.

4 Analysis of Fujisaki-Okamoto Transforms

In the following, we analyze complete non-malleability of the FO transforms. To do that, we will consider the modular treatment of the FO transforms pursued by [15]. Each FO transform is an application of two transformations, namely T and U . T takes as input an IND-CPA/OW-CPA secure PKE scheme $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ and a random oracle G , and outputs a deterministic OW-PCVA PKE scheme $\Pi_1 = (\text{Gen}_1, \text{Enc}_1, \text{Dec}_1)$ (when Π is OW-CPA, it should also satisfy γ -uniformity for a sufficiently large γ). The encryption algorithm of the transformed PKE runs the encryption algorithm of the underlying IND-CPA secure PKE scheme Π but uses $G(m)$ as its randomness, i.e.

$\text{Enc}_1(\text{pk}, m) := \text{Enc}(\text{pk}, m; \mathbf{G}(m))$. The decryption algorithm runs the decryption algorithm of the underlying PKE scheme, i.e. $m' := \text{Dec}(\text{sk}, c)$, and returns \perp if $m' = \perp$ or the re-encryption of m under public-key pk and randomness $\mathbf{G}(m)$ does not match with c , i.e. if $\text{Enc}(\text{pk}, m'; \mathbf{G}(m')) \neq c$.

Given a PKE scheme satisfying OW-PCVA or some different flavour of one-wayness (depending on the transformation we are going to use) and a random oracle \mathbf{H} , four variant of the transformation \mathbf{U} can be used to produce an IND-CCA2 KEM scheme. In fig Fig. 1 we recall the algorithms needed to instantiate the the \mathbf{U}^\perp and \mathbf{U}^\times transformations. The algorithms for the \mathbf{U}_m^\perp and \mathbf{U}_m^\times transformations are the same of \mathbf{U}^\perp and \mathbf{U}^\times respectively, except that the encapsulation algorithm computes the key \mathbf{K} as $\mathbf{H}(m)$, and the decapsulation algorithm outputs $\mathbf{K} := \mathbf{H}(m)$ when $m \neq \perp$.

<p>Algorithm $\text{Gen}^\times(1^\lambda)$</p> <hr/> <p>$(\text{pk}', \text{sk}') \leftarrow_{\\$} \text{Gen}_1(1^\lambda)$ $s \leftarrow_{\\$} \mathcal{M}$ $\text{sk} := (\text{sk}', s)$ return (pk', sk)</p> <p>Algorithm $\text{Decaps}^\perp(\text{sk}, c)$</p> <hr/> <p>$m' := \text{Dec}_1(\text{sk}, c)$ if $m' = \perp$ return \perp return $\mathbf{K} := \mathbf{H}(m', c)$</p>	<p>Algorithm $\text{Encaps}(\text{pk})$</p> <hr/> <p>$m \leftarrow_{\\$} \mathcal{M}$ $c \leftarrow_{\\$} \text{Enc}_1(\text{pk}, m)$ $\mathbf{K} := \mathbf{H}(m, c)$ return (\mathbf{K}, c)</p> <p>Algorithm $\text{Decaps}^\times(\text{sk}, c)$</p> <hr/> <p>Parse $\text{sk} = (\text{sk}', s)$ $m' := \text{Dec}_1(\text{sk}', c)$ if $m' = \perp$ return $\mathbf{K} := \mathbf{H}(m', s)$ return $\mathbf{K} := \mathbf{H}(m', c)$</p>
--	--

Fig. 1. Algorithms needed by the transformations $\mathbf{U}^\perp[\Pi_1, \mathbf{H}] = (\text{Gen}_1, \text{Encaps}, \text{Decaps}^\perp)$ and $\mathbf{U}^\times[\Pi_1, \mathbf{H}] = (\text{Gen}^\perp, \text{Encaps}, \text{Decaps}^\times)$.

In Sect. 4.1, we will analyze the $\mathbf{U}_m^\perp/\mathbf{U}_m^\times$ transformations and show that they are not completely non-malleable by giving an attack that can be performed when the underlying OW-PCVA PKE scheme is obtained by applying the transformation \mathbf{T} to a widely known IND-CPA PKE scheme.

Then, in Sect. 4.2, we will analyze the $\mathbf{U}^\perp/\mathbf{U}^\times$ transformations and show that they are not completely non-malleable by constructing a contrived OW-PCVA PKE scheme that, when given as input to $\mathbf{U}^\perp/\mathbf{U}^\times$, leads to a KEM whose NM-ATK* security can be easily broken. Then, we show that it suffices to assume a very natural property of the underlying OW-PCVA PKE scheme, named public-key uniqueness, to achieve complete non-malleability.

Finally, in Sect. 4.3, we show that it is possible to achieve a completely non-malleable KEM without assuming public-key uniqueness of the underlying PKE scheme with a little tweak to $\mathbf{U}^\perp/\mathbf{U}^\times$.

4.1 Analysis of the $U_m^{\perp/\neq}$ Transformations

In the following, we will show that U_m^{\perp} and U_m^{\neq} transformations lead to a KEM that is not completely non-malleable. In particular, we show a concrete attack that can be carried out to both $\tilde{H}_m^{\perp} := U_m^{\perp}[\mathbb{T}[\Pi, \mathbb{G}], \mathbb{H}]$ and $\tilde{H}_m^{\neq} := U_m^{\neq}[\mathbb{T}[\Pi, \mathbb{G}], \mathbb{H}]$ even against the weaker notion of NM-CPA*. Let Π be the El-Gamal encryption scheme.

When running the experiment $\text{KEM}_{\tilde{H}_m^{\perp}, \mathbb{A}}^{nm-cpa*}(\lambda)$ with \tilde{H}_m^{\perp} (resp. \tilde{H}_m^{\neq}), an efficient adversary \mathbb{A} receives as input a public key $\text{pk}^* = (\text{params}, h)$, the challenge ciphertext c^* , and an encapsulation key K^* . The challenge ciphertext c^* is computed as $(c_1^* = g^r, c_2^* = h^r \cdot m^*)$ with $r = \mathbb{G}(m^*)$, $\text{params} = (\mathbb{G}, g, q)$, where g is the generator of a cyclic group \mathbb{G} of order q , $h = g^x$ for $x \leftarrow_s \mathbb{Z}_q$, and m^* is a randomly sampled message from \mathbb{G} . The challenge key K^* is either the output of the encapsulation algorithm on input m^* (i.e., $K^* = \mathbb{H}(m^*)$), or it is sampled from the uniform distribution over the key space \mathcal{K} .

Now, \mathbb{A} can craft a new public key $\text{pk}' = (\text{params}, h \cdot g^{x'})$, for $x' \leftarrow_s \mathbb{Z}_q$, and a ciphertext $c' = (c_1^*, c_2^* \cdot c_1^{x'})$. It is straightforward to see that, since $\text{Enc}(\text{pk}', m') = (g^r, g^{(x+x')r} m^*) = c'$, there exist a key K' such that $(c', K') = \text{Encaps}(\text{pk}')$, where $c' = \text{Enc}(\text{pk}', m^*)$. Finally, since $K' := \mathbb{H}(m')$ with $m' = m^*$, then $K' = K^*$.

Now, \mathbb{A} can define the relation between pk^* and pk' as follows:

$$R_{\text{pk}}(\text{pk}^* = (\text{params}, h), \text{pk}' = (\text{params}, h')) = 1 \text{ iff } h' = h \cdot g^{x'}, x' \in \mathbb{Z}_p.$$

Moreover, \mathbb{A} can define the relation between K^* and K' as the identity relation, i.e. $R_K(K^*, K') = 1$ iff $K^* = K'$. As shown above, \mathbb{A} can come up with a $\text{pk}' \neq \text{pk}$ satisfying the relation $R(K^*, K', \text{pk}^*, \text{pk}, c') = R_K(K^*, K') \wedge R_{\text{pk}}(\text{pk}^*, \text{pk}')$. Such relation, when the ciphertext c' is computed as above (i.e. $c' = (c_1^*, c_2^* \cdot c_1^{x'})$), leads the $\text{KEM}_{\tilde{H}_m^{\perp}, \mathbb{A}}^{nm-cpa*}(\lambda)$ experiment output 1 and the $\text{KEM}_{\tilde{H}_m^{\perp}, \mathbb{A}, \mathbb{S}}^{nm-cpa*}(\lambda)$ output 0 with non-negligible probability.

In the following, we will show that the $U^{\perp/\neq}$ transformations can be turned into a completely non-malleable KEM without much effort.

4.2 Analysis of the $U^{\perp/\neq}$ Transformations

The U^{\perp} and U^{\neq} transformations do not naturally satisfy complete non-malleability. Indeed, given an OW-PCVA PKE scheme $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$, we can construct a contrived OW-PCVA PKE scheme $\Pi' = (\text{Gen}', \text{Enc}', \text{Dec}')$ that is trivially insecure against the NM-CPA* experiment. The scheme Π' follows:

<p>Algorithm $\text{Gen}'(1^\lambda)$</p> <p>$(\text{pk}, \text{sk}) \leftarrow_s \text{Gen}(1^\lambda)$</p> <p>$b \leftarrow_s \{0, 1\}$</p> <p>$\text{pk}' := \text{pk} b$</p> <p>return (pk', sk)</p>	<p>Algorithm $\text{Enc}'(\text{pk}', m)$</p> <p>Parse $\text{pk}' = (\text{pk}, b)$</p> <p>$c \leftarrow_s \text{Enc}(\text{pk}, m)$</p> <p>return c</p>	<p>Algorithm $\text{Dec}'(\text{sk}, m)$</p> <p>$m := \text{Dec}(\text{sk}, m)$</p> <p>return m</p>
--	---	--

It is straightforward to see that the adversary of NM-CPA*, when run with $\tilde{\Pi}^\perp := \mathbf{U}^\perp[\Pi', \mathbf{H}]$, distinguishes between the two experiments with non-negligible probability. Indeed, an adversary \mathbf{A} can come up with a public key \mathbf{pk}' different of the public key generated by $\tilde{\Pi}^\perp$ that leads to the same ciphertext c^* encapsulating \mathbf{K}^* as computed in the $\text{KEM}_{\tilde{\Pi}^\perp, \mathbf{A}}^{\text{nm-cpa}^*}$ experiment. To be precise, the adversary \mathbf{A} might define a relation $R(\mathbf{K}^*, \mathbf{K}', \mathbf{pk}^*, \mathbf{pk}, \cdot) = R_{\mathbf{K}}(\mathbf{K}^*, \mathbf{K}') \wedge R_{\mathbf{pk}}(\mathbf{pk}^*, \mathbf{pk}')$, where $R_{\mathbf{K}}(\mathbf{K}^*, \mathbf{K}')$ is the identity function and $R_{\mathbf{pk}}(\mathbf{pk}^*, \mathbf{pk}') = 1$ iff, given that \mathbf{pk}^* can be parsed as (\mathbf{pk}, b) with $b \in \{0, 1\}$, then \mathbf{pk}' equals $(\mathbf{pk}, 1 - b)$. If c^* is the encryption of a message m^* under \mathbf{pk}^* , then it trivially encrypts m^* under \mathbf{pk}' . Hence, the key $\mathbf{K}^* := \mathbf{H}(m^*, c^*)$ will be the same.

An idea to avoid such an artificial attack, is to restrict the set of admitted PKE schemes to the ones for which such attack is not possible to carry out in the first place. As we will show, this suffice to guarantee complete non-malleability of $\tilde{\Pi}^\perp$. To do that, we introduce a natural property called public-key uniqueness, informally stating that it is infeasible for an adversary to come up with two different public keys leading to the same ciphertext when encrypting the same message. For example, in El-Gamal, the encryption of the same message with two different public keys will always lead to a different ciphertext for any possible random coins and any possible message. This property, which we formalize below, is indeed achieved by most of the known PKE schemes.

Definition 9 (Public-Key Uniqueness). *A public-key encryption scheme $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ with message space \mathcal{M} is public-key unique if, for all unbounded/PPT adversary \mathbf{A} , there exists a negligible function $\text{negl}(\lambda)$ such that*

$$\Pr \left[\exists r, r', m : \begin{array}{c} \text{Enc}(\mathbf{pk}, m; r) = \text{Enc}(\mathbf{pk}', m; r') \\ \wedge (\mathbf{pk} \neq \mathbf{pk}') \end{array} \mid (\mathbf{pk}, \mathbf{pk}') \leftarrow_{\mathbf{A}} \mathbf{A}(1^\lambda) \right] \leq \text{negl}(\lambda).$$

We say that Π is perfect public-key unique if do not exist two public keys \mathbf{pk} and \mathbf{pk}' with $\mathbf{pk}' \neq \mathbf{pk}$ such that $\text{Enc}(\mathbf{pk}, m; r) = \text{Enc}(\mathbf{pk}', m; r')$ for any $m \in \mathcal{M}$ and any randomness r, r' .

In the following, we show that the KEM $\tilde{\Pi}^\perp = \mathbf{U}^\perp[\Pi_1, \mathbf{H}]$ is completely non-malleable when the PKE Π_1 is OW-PCVA and satisfies perfect public-key uniqueness. Note that our result trivially extends when Π_1 satisfies public-key uniqueness against PPT or unbounded adversaries.

Theorem 2 (Π_1 pk-unique OW-PCVA $\xrightarrow{\text{ROM}}$ $\tilde{\Pi}^\perp$ NM-CCA2*). *Assuming the existence of a random oracle \mathbf{H} , if Π_1 is a correct OW-PCVA secure PKE (Definition 3) satisfying perfect public-key uniqueness (Definition 9), then $\tilde{\Pi}^\perp$ defined as above is a correct NM-CCA2* secure KEM (Definition 8).*

Proof. Correctness of $\tilde{\Pi}^\perp$ trivially follows from the correctness of Π_1 . The idea behind the proof is to simulate the decapsulation oracle without using the secret key. We can do that by answering the decapsulation queries with a random key, and next simulate the random oracle \mathbf{H} by using the plaintext checking oracle $\text{PCO}(\text{sk}^*, \cdot, \cdot)$, provided by the OW-PCVA game. Furthermore, we will

use the ciphertext validity oracle $\text{CVO}^{(c^*)}(\text{sk}^*, \cdot)$ in order to reject decapsulation queries for invalid ciphertexts. Before proceeding with the proof we need to make the following observations. Since a NM-CCA2* adversary against \tilde{H}^\perp is allowed to choose both the public key and the ciphertext, the decapsulation oracle can receive ciphertexts encrypted using a public key $\text{pk}' \neq \text{pk}^*$. However, since the decapsulation oracle does not know the secret key associated to pk' , we cannot require to it to check the validity of such ciphertexts. In other words, the decapsulation oracle is only required to check the validity of ciphertexts encrypted using the challenge public key pk^* . A similar observation shall be done for the oracle H. Due to the fact that also H can receive ciphertexts encrypted using any public key distinct from pk^* , we should be able to check the validity of such ciphertexts. In other words, given a message m and a ciphertext c , we should allow H to check whether $m = \text{Dec}_1(\text{sk}', c)$. However, since H doesn't know the secret key associated with pk' , this desired behavior cannot be achieved. For this reason, the same approach used for $\text{Decaps}_\perp^{(c^*)}(\text{sk}^*, \cdot)$ applies. Let B be a PPT adversary that breaks NM-CCA2* security of \tilde{H}^\perp with non-negligible probability issuing a polynomially bounded number of queries to Decaps_\perp and H. The sequence of games we are going to consider is described in Fig. 2.

Lemma 1. $\text{KEM}_{\tilde{H}^\perp, \text{B}}^{\text{nm-cca2}^*}(\lambda) \equiv \mathbf{G}_0^{\text{B}}(\lambda)$.

Proof. Let us start by noticing that in game \mathbf{G}_0 the challenger takes a uniform message m^* , computes $c^* \leftarrow \text{Enc}_1(\text{pk}^*, m^*)$, $K^* := \text{H}(m^*, c^*)$ and outputs (pk^*, c^*, K^*) . This game coincides exactly with the left experiment of the NM-CCA2* definition. Thus, $\text{KEM}_{\tilde{H}^\perp, \text{B}}^{\text{nm-cca2}^*}(\lambda) \equiv \mathbf{G}_0^{\text{B}}(\lambda)$. \square

Lemma 2. $\mathbf{G}_0^{\text{B}}(\lambda) \equiv \mathbf{G}_1^{\text{B}}(\lambda)$.

Proof. Differently from \mathbf{G}_0 , in game \mathbf{G}_1 we have modified the oracles $\text{Decaps}_\perp^{(c^*)}$ and H in order to avoid the usage of the secret key. In particular, \mathbf{G}_1 defines two sets \mathcal{L}_H and \mathcal{L}_D , where \mathcal{L}_H contains all entries of the form (m, c, K) when either $\text{Decaps}_\perp^{(c^*)}$ is queried about a ciphertext c or H was queried about (m, c) , and \mathcal{L}_D contains all the entries (c, K) when either $\text{Decaps}_\perp^{(c^*)}$ is queried about c or H is queried about (m, c) for $m = \text{Dec}_1(\text{sk}', c)$. Now, we want to show that the view of B in \mathbf{G}_0 and \mathbf{G}_1 is distributed exactly in the same manner. For this purpose, let us consider a ciphertext c' and a message $m' = \text{Dec}_1(\text{sk}', c')$ for which B has never been queried $\text{Decaps}_\perp^{(c^*)}$:

- **Case $m' = \perp$:** in game \mathbf{G}_0 , $\text{Dec}_1(\text{sk}', c')$ will return \perp to indicate that c' is a malformed ciphertext, which is exactly the behavior the $\text{Decaps}_\perp^{(c^*)}$ has in game \mathbf{G}_1 . Regarding the behavior of H, we can see that in both games H returns a randomly chosen key.

Games $\mathbf{G}_0^B(\lambda) - \mathbf{G}_3^B(\lambda)$

$(pk^*, sk^*) \leftarrow_s \text{Gen}_1(1^\lambda)$
 $st \leftarrow_s \text{B}_1^{\text{Decaps}_\perp(sk^*, \cdot), \text{H}(\cdot, \cdot)}(pk^*)$
 $m^* \leftarrow_s \mathcal{M}$
 $c^* \leftarrow_s \text{Enc}_1(pk, m^*)$
 $K^* := \text{H}(m^*, c^*) \quad \boxed{\#G_0, G_1}$
 $K^* \leftarrow_s \{0, 1\}^\lambda \quad \boxed{\#G_2, G_3}$
 $(pk', R, c') \leftarrow_s \text{B}_2^{\text{Decaps}_\perp^{(c^*)}(sk^*, \cdot), \text{H}(\cdot, \cdot)}(pk^*, c^*, K^*, st)$
return 1 iff $\exists(K', r)$ such that
 $(c', K') := \text{Encaps}(pk'; r) \wedge (K' \neq \perp)$
 $\wedge (pk' \neq pk^* \vee K' \neq K^*)$
 $\wedge R(K', K^*, pk', pk^*, c')$

Oracle $\text{Decaps}_\perp^{(c^*)}(sk^*, c) \quad \boxed{\#G_0, G_3}$

$m' := \text{Dec}_1(sk^*, c)$
if $m' = \perp$
 return \perp
return $K := \text{H}(m', c)$

Oracle $\text{H}(m, c)$

if $\exists K$ s.t. $(m, c, K) \in \mathcal{L}_H$
 return K
 $K \leftarrow_s \mathcal{K}$
if $\text{Dec}_1(sk^*, c) = m \quad \boxed{\#G_1, G_2}$
 if $c = c^* \quad \boxed{\#G_2}$
 return $\perp \quad \boxed{\#G_2}$
 if $\exists K'$ s.t. $(c, K') \in \mathcal{L}_D \quad \boxed{\#G_1}$
 $K := K' \quad \boxed{\#G_1}$
 else $\boxed{\#G_1}$
 $\mathcal{L}_D := \mathcal{L}_D \cup \{(c, K)\} \quad \boxed{\#G_1}$
 $\mathcal{L}_H := \mathcal{L}_H \cup \{(m, c, K)\}$
return K

Oracle $\text{Decaps}_\perp^{(c^*)}(sk^*, c) \quad \boxed{\#G_1, G_2}$

if $\exists K$ s.t. $(c, K) \in \mathcal{L}_D$
 return K
 $K \leftarrow_s \mathcal{K}$
 $m' := \text{Dec}_1(sk^*, c)$
if $m' = \perp$
 $K := \perp$
 $\mathcal{L}_D := \mathcal{L}_D \cup \{(c, K)\}$
 $\mathcal{L}_H := \mathcal{L}_H \cup \{(m', c, K)\}$
return K

Fig. 2. Sequence of games needed to prove Theorem 2 and the consequential oracle modifications.

- **Case** $m' \neq \perp$: in game \mathbf{G}_0 , $\text{Decaps}_\perp^{(c^*)}$ returns $K := \text{H}(m', c')$ which is either a fresh key randomly chosen from the key space if (m', c') has never been queried to H , or taken from \mathcal{L}_H if (m', c') was already stored in \mathcal{L}_H . In game \mathbf{G}_1 we need to consider two sub-cases:
 - **B** first queries H about (m', c') and then queries $\text{Decaps}_\perp^{(c^*)}$ about c' : In this case, H returns a key K which is either a fresh key randomly chosen from the key space, or it is already stored in \mathcal{L}_H . Since $\text{Decaps}_\perp^{(c^*)}$ has not been queried about c' yet, H will add (c, K) to \mathcal{L}_D . Next, when $\text{Decaps}_\perp^{(c^*)}$ will be queried about c' , it will return a key K stored in \mathcal{L}_D that coincides with the key stored in \mathcal{L}_H , as in game \mathbf{G}_0 .

- \mathcal{B} first queries $\text{Decaps}_{\perp}^{(c^*)}$ about c' and then queries H about (m', c') : In this case, $\text{Decaps}_{\perp}^{(c^*)}$ returns a randomly chosen K , which is added to both \mathcal{L}_D and \mathcal{L}_H . Subsequently, when H is queried about (m', c') , the oracle will return the key K stored in \mathcal{L}_H that coincides with the key K stored in \mathcal{L}_D . This ensures that $\text{Decaps}_{\perp}^{(c^*)}(c') = \text{H}(m', c') = K$, as in game \mathbf{G}_0 .

Therefore, we have that $\mathbf{G}_1^{\mathbf{B}}(\lambda) \equiv \mathbf{G}_0^{\mathbf{B}}(\lambda)$. □

Lemma 3. $\mathbf{G}_1^{\mathbf{B}}(\lambda) \approx_c \mathbf{G}_2^{\mathbf{B}}(\lambda)$.

Proof. In game \mathbf{G}_2 we make the following two modifications. First, the challenger takes a uniformly sampled key rather than the real key output by the oracle H , and second we make the oracle H output \perp when queried about (m^*, c^*) . By denoting the latter event QUERY , we notice that \mathbf{G}_1 and \mathbf{G}_2 are identically distributed conditioned to the event QUERY not happening. Thus, the only hope for the adversary to distinguish between the \mathbf{G}_1 and \mathbf{G}_2 is to trigger the QUERY event in \mathbf{G}_2 . It is easy to see that when \mathbf{B} queries H in game \mathbf{G}_1 about (m^*, c^*) , H will return the challenge encapsulation key K^* . Instead, in game \mathbf{G}_2 , the game returns \perp when \mathbf{B} queries H about (m^*, c^*) . Let us now assume that QUERY is not triggered. Since K^* is either an output of H or a randomly chosen value, the adversary can only try to distinguish by guessing the plaintext m^* of c^* , calculate $K' := \text{H}(m^*, c^*)$ and then check whether K' is equal to K^* . However, this coincides with the QUERY event. Alternatively, the adversary might try to distinguish by making \mathbf{G}_1 always output 1, i.e. \mathbf{B} may try to come up with a tuple (pk', R, c') for which the relation $R(K', K^*, pk', pk^*, c')$ holds for a key K^* produced by the encapsulation algorithm under pk' (as in \mathbf{G}_1), but does not hold when such key is randomly chosen (as in \mathbf{G}_2). Note that H is a random oracle, so the set of possible relations is restricted to the ones having $K' = K^* = \text{H}(m^*, c^*)$, otherwise even a single bit different than (m^*, c^*) in the input of H leads to an independent output. For such a relation the adversary may try to find a public key pk' for which the encapsulation of K^* under pk' leads to c^* . However, for the perfect public-key uniqueness property of Π_1 , such public key does not exist. Since K' can be computed only by giving as input to H a pair (m', c') in which either $m' \neq m^*$ or $c' \neq c^*$, such a key is independent from the challenge key. Hence, the distribution of the adversary's view in both games is identical given that QUERY does not happen.

Now, to estimate $\Pr[\text{QUERY}]$ we construct an efficient adversary \mathbf{A} breaking OW-PCVA of Π_1 when QUERY occurs. In particular, we define \mathbf{A} in Fig. 3. Notice that \mathbf{A} perfectly simulates \mathbf{G}_1 . Indeed, the occurrence of QUERY implies that \mathbf{B} has queried H about (m, c) , in which $(m, c) \in \mathcal{L}_H$ for $m = m^*$ and $c = c^*$. \mathbf{A} will return $m = m^*$ and win the OW-PCVA experiment. Since such condition coincides with the QUERY event, we get that the probability of \mathbf{B} of triggering QUERY coincides with the probability of \mathbf{A} in winning the OW-PCVA experiment, i.e. $\Pr[\text{QUERY}] = \Pr[\text{PKE}_{\Pi_1, \mathbf{A}}^{\text{ow-pcva}}(\lambda)]$. □

<p>Adversary $A^{\text{PCO}(\text{sk}^*, \cdot, \cdot), \text{CVO}^{(c^*)}(\text{sk}^*, \cdot)}(\text{pk}, c^*)$</p> <hr style="border: 0.5px solid black;"/> <p>$K^* \leftarrow_s \mathcal{K}$</p> <p>$(\text{pk}', R, c') \leftarrow_s B_2^{\text{Decaps}_{\perp}^{(c^*)}(\cdot), H(\cdot, \cdot)}(\text{pk}^*, c^*, K^*)$</p> <p>if $\exists(m, c^*) \in \mathcal{L}_H$ s.t. $\text{PCO}(\text{sk}^*, m, c^*) = 1$</p> <p style="padding-left: 20px;">return m</p> <p>return \perp</p> <p>Oracle $\text{Decaps}_{\perp}^{(c^*)}(\text{sk}, c)$</p> <hr style="border: 0.5px solid black;"/> <p>if $\exists c$ s.t. $(c, K) \in \mathcal{L}_D$</p> <p style="padding-left: 20px;">return K</p> <p>if $\text{CVO}^{(c^*)}(\text{sk}^*, c) = 0$</p> <p style="padding-left: 20px;">return \perp</p> <p>$K \leftarrow_s \mathcal{K}$</p> <p>$\mathcal{L}_D := \mathcal{L}_D \cup \{(c, K)\}$</p> <p>return K</p>	<p>Oracle $H(m, c)$</p> <hr style="border: 0.5px solid black;"/> <p>if $\exists K$ s.t. $(m, c, K) \in \mathcal{L}_H$</p> <p style="padding-left: 20px;">return K</p> <p>$K \leftarrow_s \mathcal{K}$</p> <p>if $\text{PCO}(\text{sk}^*, m, c) = 1$</p> <p style="padding-left: 20px;">if $\exists K'$ s.t. $(c, K') \in \mathcal{L}_D$</p> <p style="padding-left: 40px;">$K := K'$</p> <p style="padding-left: 20px;">else</p> <p style="padding-left: 40px;">$\mathcal{L}_D := \mathcal{L}_D \cup \{(c, K)\}$</p> <p>$\mathcal{L}_H := \mathcal{L}_H \cup \{(m, c)\}$</p> <p>return K</p>
---	---

Fig. 3. Adversary A breaking security of the underlying OW-PCVA PKE.

Lemma 4. $\mathbf{G}_2^{\mathbf{B}}(\lambda) \approx_c \mathbf{G}_3^{\mathbf{B}}(\lambda)$.

Proof. Notice that game \mathbf{G}_3 is identically distributed to \mathbf{G}_0 , except that the challenge encapsulation key is randomly chosen from the key space. We will show that the view of B in \mathbf{G}_2 and \mathbf{G}_3 is identically distributed, under the condition that the QUERY event does not occur.

Let us fix a ciphertext c' and a message $m' = \text{Dec}_2(\text{sk}^*, c')$. We consider two cases:

- **Case** $m' \notin \mathcal{M}$: in game \mathbf{G}_3 , when $\text{Decaps}_{\perp}^{(c^*)}$ is queried about c' , it will return \perp , which is exactly what $\text{Decaps}_{\perp}^{(c^*)}$ returns in \mathbf{G}_2 .
- **Case** $m' \in \mathcal{M}$. Here, we need to consider two sub-cases:
 - B first queries $\text{Decaps}_{\perp}^{(c^*)}$ and then H: assume that neither $\text{Decaps}_{\perp}^{(c^*)}$ nor H have been queried before about c' and (m', c') respectively. In \mathbf{G}_2 , when B queries $\text{Decaps}_{\perp}^{(c^*)}$, $\text{Decaps}_{\perp}^{(c^*)}$ will return a uniform key K , add an entry of the form (c', K) to \mathcal{L}_D and an entry of the form (m', c', K) to \mathcal{L}_H . Next, when B queries H about (m', c', pk') , H will return the same key returned by $\text{Decaps}_{\perp}^{(c^*)}$ since $(m', c', K) \in \mathcal{L}_H$. In \mathbf{G}_3 , $\text{Decaps}_{\perp}^{(c^*)}$ will return a uniform key $K = H(m', c')$. Next, when B queries H about (m', c') , H will return the same key K . Therefore, in both games we have $\text{Decaps}_{\perp}^{(c^*)} = K = H(m', c')$.
 - B first queries H and then $\text{Decaps}_{\perp}^{(c^*)}$: As before, let us assume that neither $\text{Decaps}_{\perp}^{(c^*)}$ nor H have been queried before about c' and (m', c')

respectively. In \mathbf{G}_2 , when \mathbf{B} queries \mathbf{H} , \mathbf{H} will return a uniform key \mathbf{K} and add an entry of the form (c', \mathbf{K}) to \mathcal{L}_D . Next, when $\text{Decaps}_{\perp}^{(c^*)}$ is queried about c' , $\text{Decaps}_{\perp}^{(c^*)}$ will return the same key returned by \mathbf{H} , since $(c', \mathbf{K}) \in \mathcal{L}_D$. In game \mathbf{G}_3 when \mathbf{B} queries \mathbf{H} , \mathbf{H} will return a uniform key \mathbf{K} and add an entry of the form (m', c', \mathbf{K}) to \mathcal{L}_H . Next, when $\text{Decaps}_{\perp}^{(c^*)}$ is queried about c' , $\text{Decaps}_{\perp}^{(c^*)}$ will return the same key \mathbf{K} , due to the fact that $(m', c', \mathbf{K}) \in \mathcal{L}_H$. Thus, in both games we have $\mathbf{H}(m', c') = \mathbf{K} = \text{Decaps}_{\perp}^{(c^*)}$.

Since the only hope for the adversary is to trigger the QUERY event in \mathbf{G}_3 , the probability for \mathbf{B} to distinguish between \mathbf{G}_2 and \mathbf{G}_3 is bounded by the probability of winning the OW-PCVA game of the underlying PKE scheme. \square

Lemma 5. $\mathbf{G}_3^{\mathbf{B}}(\lambda) \equiv \text{KEM}_{\tilde{\Pi}_1, \mathbf{B}, \$}^{nm-cca2*}(\lambda)$.

Proof. Since \mathbf{G}_3 is similar to \mathbf{G}_0 , with the only difference that the encapsulation key \mathbf{K}^* is uniform and independent from the one obtained by querying \mathbf{H} , for the same considerations that we did for \mathbf{G}_0 , it holds that the two distributions are identically distributed. \square

Combining the above lemmas, we get that $\text{KEM}_{\tilde{\Pi}_1, \mathbf{B}}^{nm-cca2*}(\lambda) \approx_c \text{KEM}_{\tilde{\Pi}_1^+, \mathbf{B}, \$}^{nm-cca2*}(\lambda)$. \square

4.3 Modified Transformation $\hat{\mathbf{U}}^{\perp}$

In the following, we leverage the idea of prefix hashing introduced by Duman et al. [10], to construct a completely non-malleable KEM without requiring public-key uniqueness of the underlying PKE. In particular, our $\hat{\mathbf{U}}^{\perp}$ is identical to \mathbf{U}^{\perp} , except that now the encapsulation algorithm gives as input to the random oracle \mathbf{H} also the public key \mathbf{pk} together with the message m and the ciphertext c from the underlying PKE scheme, i.e. $\mathbf{K} := \mathbf{H}(m, c, \mathbf{pk})$. Note that now the decapsulation oracle must take as input also the challenge public key together with the challenge secret key in order to recompute $\mathbf{H}(m, c, \mathbf{pk})$. The theorem below states that $\tilde{\Pi}_1^{\perp} := \hat{\mathbf{U}}^{\perp}[\Pi_1, \mathbf{H}]$ is completely non-malleable. Since the techniques used to prove the theorem below are similar to the ones used to prove Theorem 2, we will only highlight the changes in the sequence of games w.r.t. the proof of Theorem 2, and the changes needed to prove some lemma when required.

Theorem 3 (Π_1 OW-PCVA \xrightarrow{ROM} $\tilde{\Pi}_1^{\perp}$ NM-CCA2*). *Assuming the existence of a random oracle \mathbf{H} , if Π_1 is a correct OW-PCVA secure PKE (Definition 3), then $\tilde{\Pi}_1^{\perp}$ defined above is a correct NM-ATK* secure KEM (Definition 8).*

The proof of the theorem appears in Appendix A.2.

5 Relation with Completely Non-malleable PKE

In the following, we will show which kind of relationships exist between a NM-ATK* PKE schemes and NM-ATK* KEM schemes. We will proceed by first showing that a NM-ATK* PKE scheme is intrinsically a NM-ATK* KEM scheme, and next we will introduce a construction to prove that a NM-ATK* KEM can be used together with a NM-ATK SKE scheme to construct a NM-ATK* PKE scheme by using the KEM/DEM paradigm.

5.1 NM-ATK* PKE \implies NM-ATK* KEM

Wlog, we can assume that $\mathcal{M} = \mathcal{K}$. Let $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ be a PKE scheme, and $\Pi' = (\text{Gen}, \text{Encaps}, \text{Decaps})$ be a KEM scheme defined as follows:

<p style="text-align: center;"><u>Algorithm Encaps(pk)</u></p> <p>$K \leftarrow \mathcal{K}$</p> <p>$c \leftarrow \text{Enc}(\text{pk}, K)$</p> <p>return (c, K)</p>	<p style="text-align: center;"><u>Algorithm Decaps(sk, c)</u></p> <p>$K := \text{Dec}(\text{sk}, c)$</p> <p>return K</p>
--	--

Theorem 4 (NM-ATK* PKE \implies NM-ATK* KEM). *If Π is a NM-ATK* secure PKE (Definition 4) with respect to a set of relations \mathcal{R} , then Π' is a NM-ATK* secure KEM (Definition 8) with respect to \mathcal{R} .*

The proof of the theorem appears in Appendix A.3.

5.2 NM-ATK* KEM + NM-ATK SKE \implies NM-ATK* PKE

It is well known that a KEM scheme alone doesn't allow to build a PKE scheme, due to the fact that an encapsulation algorithm can be instantiated by encrypting a uniformly chosen message. To solve this issue, a secret-key encryption scheme can be used along with a key-encapsulation mechanism.

Let $\Pi^{kem} = (\text{Gen}^{kem}, \text{Encaps}, \text{Decaps})$ be a NM-ATK* KEM with key space \mathcal{K} , and $\Pi^{ske} = (\text{Gen}^{ske}, \text{Enc}, \text{Dec})$ be a NM-ATK SKE scheme with message space \mathcal{M} and the same key space \mathcal{K} of Π^{kem} , we can construct a hybrid PKE scheme $\Pi^{hy} = (\text{Gen}^{hy}, \text{Enc}^{hy}, \text{Dec}^{hy})$ as defined below.

<p style="text-align: center;"><u>Algorithm Gen^{hy}(1^λ)</u></p> <p>$(\text{pk}, \text{sk}) \leftarrow \text{Gen}^{kem}(1^\lambda)$</p> <p>return (pk, sk)</p>	<p style="text-align: center;"><u>Algorithm Enc^{hy}(pk, m)</u></p> <p>$(c, K) \leftarrow \text{Encaps}(\text{pk})$</p> <p>$c' \leftarrow \text{Enc}(K, m)$</p> <p>return (c, c')</p>	<p style="text-align: center;"><u>Algorithm Dec^{hy}(sk, c)</u></p> <p>$K := \text{Decaps}(\text{sk}, c)$</p> <p>return $m := \text{Dec}(K, c)$</p>
--	---	--

We prove that if Π^{kem} is a completely non-malleable KEM and Π^{ske} is a non-malleable SKE, then Π^{hy} is a completely non-malleable PKE.

Theorem 5. *If Π^{kem} is an NM-ATK* secure KEM (Definition 8) with respect to a set of relations \mathcal{R} and Π^{ske} is an NM-ATK secure SKE (Definition 6) with respect to a set of relations $\mathcal{R}' \subseteq \mathcal{R}$, then the scheme Π^{hy} described above is a NM-ATK* secure PKE scheme (Definition 4) with respect to \mathcal{R} .*

The proof of the theorem appears in Appendix A.4.

Acknowledgements. We thank all the anonymous reviewer for their insightful comments. The work of the first and third authors is partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU and by Sapienza University under the project SPECTRA.

A Supporting Proofs

A.1 Proof of Theorem 1

Proof. The intuition behind the proof is to define the scheme Π' in such a way that the adversary can leverage its structure to succeed in the NM-ATK* experiment. In particular, we define Π' in the following way:

Algorithm $\text{Gen}'(1^\lambda)$	Algorithm $\text{Encaps}'(\text{pk}')$	Algorithm $\text{Decaps}'(\text{sk}, c)$
$(\text{pk}, \text{sk}) = \text{Gen}(1^\lambda)$	Parse pk' as $\text{pk}' := \text{pk} b$	$\text{K} := \text{Decaps}'(\text{sk}, c)$
$b \leftarrow_{\$} \{0, 1\}$	$(c, \text{K}) \leftarrow_{\$} \text{Encaps}(\text{pk})$	
$\text{pk}' := \text{pk} b$		
return (pk', sk)	return (c, K)	return K

We can clearly see that Π' is not NM-ATK* secure. Indeed, an efficient adversary \mathbf{A} receiving a public key pk , the challenge ciphertext c and an encapsulation key K (either the real or the fake one), just have to flip the last bit of pk denoted $\text{pk}' := \text{pk}||\bar{b}$, a relation R and the challenge ciphertext c . In this case, \mathbf{A} will always succeeds in breaking NM-ATK* security of Π' . However, Π' is still NM-ATK secure. Indeed, this is true because the adversary has to break NM-ATK security of Π' under the key pk , but for how Π' is defined, this is equivalent to break NM-ATK security of Π . If an adversary is able to break NM-ATK security of Π' , then he can also break NM-ATK security of Π , and this represent a contradiction to our assumption that Π is NM-ATK secure. \square

A.2 Proof of Theorem 3

Proof. The sequence of games and the consequential differences in the oracles are described in Fig. 4.

The proofs that $\text{KEM}_{\Pi_1^\perp, \text{B}}^{nm-cca2*}(\lambda) \equiv \mathbf{G}_0^{\text{B}}(\lambda)$, $\mathbf{G}_0^{\text{B}}(\lambda) \equiv \mathbf{G}_1^{\text{B}}(\lambda)$, and that $\mathbf{G}_2^{\text{B}}(\lambda) \approx_c \mathbf{G}_3^{\text{B}}(\lambda)$ are identical to Lemma 1, 2 and 4 respectively.

Lemma 6. $\mathbf{G}_1^{\text{B}}(\lambda) \approx_c \mathbf{G}_2^{\text{B}}(\lambda)$.

Proof. The proof is identical to Lemma 3, except that now we do not use public key uniqueness of the underlying PKE scheme to argue that the \mathbf{G}_1 and \mathbf{G}_2 are identically distributed conditioned to the fact that the event QUERY does not happen. In this case, the adversary can try to distinguish between \mathbf{G}_1 and \mathbf{G}_2 by guessing the plaintext m^* of c^* , calculate $K' := H(m^*, c^*, \text{pk}^*)$ and then check whether K' is equal to K^* or not. However, this coincides with the QUERY event. Alternatively, the adversary might try to distinguish making \mathbf{G}_1 always output 1, i.e. \mathbf{B} tries to come up with a tuple (pk', R, c') for which the relation $R(K', K^*, \text{pk}', \text{pk}^*, c')$ holds for a key K^* encapsulated by c^* under pk' (as in \mathbf{G}_1), but does not hold when such key is randomly chosen (as in \mathbf{G}_2). However, for the random oracle assumption, since pk' is part of the input of H , the key $K' := H(m^*, c^*, \text{pk}')$ will be independent from $K^* := H(m^*, c^*, \text{pk}^*)$. Thus, the distribution of \mathbf{G}_1 and \mathbf{G}_2 is identical when QUERY is not triggered.

Games $\mathbf{G}_0^{\mathbf{B}}(\lambda) - \mathbf{G}_3^{\mathbf{B}}(\lambda)$

$(\text{pk}^*, \text{sk}^*) \leftarrow \text{Gen}_1(1^\lambda)$
 $\text{st} \leftarrow \text{B}_1^{\text{Decaps}_{\perp}^{(c^*)}(\text{sk}^*, \cdot, H(\cdot, \cdot, \cdot))}(\text{pk}^*)$
 $m^* \leftarrow \mathcal{M}$
 $c^* \leftarrow \text{Enc}_1(\text{pk}, m^*)$
 $K^* := H(m^*, c^*, \text{pk}^*) \quad \boxed{\#\mathbf{G}_0, \mathbf{G}_1}$
 $K^* \leftarrow \{0, 1\}^\lambda \quad \boxed{\#\mathbf{G}_2, \mathbf{G}_3}$
 $(\text{pk}', R, c') \leftarrow \text{B}_2^{\text{Decaps}_{\perp}^{(c^*)}(\text{sk}^*, \cdot, H(\cdot, \cdot, \cdot))}(\text{pk}^*, c^*, K^*, \text{st})$
return 1 iff $\exists (K', r)$ such that
 $(c', K') := \text{Encaps}(\text{pk}'; r) \wedge (K' \neq \perp)$
 $\wedge (\text{pk}' \neq \text{pk}^* \vee K' \neq K^*)$
 $\wedge R(K', K^*, \text{pk}', \text{pk}^*)$

Oracle $\text{Decaps}_{\perp}^{(c^*)}(\text{sk}^*, \text{pk}^*, c) \quad \boxed{\#\mathbf{G}_0, \mathbf{G}_3}$

$m := \text{Dec}_1(\text{sk}^*, c)$
if $m' = \perp$
 return \perp
return $K := H(m', c, \text{pk}^*)$

Oracle $H(m, c, \text{pk})$

if $\exists K$ s.t. $(m, c, \text{pk}, K) \in \mathcal{L}_H$
 return K
 $K \leftarrow \mathcal{K}$
if $\text{Dec}_1(\text{sk}^*, c) = m \quad \boxed{\#\mathbf{G}_1, \mathbf{G}_2}$
 if $c = c^*$
 return $\perp \quad \boxed{\#\mathbf{G}_2}$
 if $\exists K'$ s.t. $(c, K') \in \mathcal{L}_D \quad \boxed{\#\mathbf{G}_1}$
 $K := K' \quad \boxed{\#\mathbf{G}_1}$
 else $\boxed{\#\mathbf{G}_1}$
 $\mathcal{L}_D := \mathcal{L}_D \cup \{(c, K)\} \quad \boxed{\#\mathbf{G}_1}$
 $\mathcal{L}_H := \mathcal{L}_H \cup \{(m, c, \text{pk}, K)\}$
return K

Oracle $\text{Decaps}_{\perp}^{(c^*)}(\text{sk}^*, \text{pk}^*, c) \quad \boxed{\#\mathbf{G}_1, \mathbf{G}_2}$

if $\exists K$ s.t. $(c, K) \in \mathcal{L}_D$
 return K
 $K \leftarrow \mathcal{K}$
 $m' := \text{Dec}_1(\text{sk}^*, c)$
if $m' = \perp$
 $K := \perp$
 $\mathcal{L}_D := \mathcal{L}_D \cup \{(c, K)\}$
 $\mathcal{L}_H := \mathcal{L}_H \cup \{(m', c, \text{pk}^*, K)\}$
return K

Fig. 4. Sequence of games needed to prove Theorem 3 and the consequential modifications of the oracles.

<p>Adversary $A^{\text{PCO}(\text{sk}^*, \cdot, \cdot), \text{CVO}^{(c^*)}(\text{sk}^*, \cdot)}(\text{pk}^*, c^*)$</p> <hr style="border: 0.5px solid black;"/> <p>$K^* \leftarrow_s \mathcal{K}$</p> <p>$(\text{pk}', R, c') \leftarrow_s B_2^{\text{Decaps}_{\perp}^{(c^*)}(\cdot), H(\cdot, \cdot, \cdot)}(\text{pk}^*, c^*, K^*)$</p> <p>if $\exists(m, c^*, \text{pk}^*) \in \mathcal{L}_H$ s.t. $\text{PCO}(\text{sk}^*, m, c^*) = 1$</p> <p style="padding-left: 20px;">return m</p> <p>return \perp</p> <p>Oracle $\text{Decaps}_{\perp}^{(c^*)}(c)$</p> <hr style="border: 0.5px solid black;"/> <p>if $\exists(c, K) \in \mathcal{L}_D$</p> <p style="padding-left: 20px;">return K</p> <p>if $\text{CVO}^{(c^*)}(\text{sk}^*, c) = 0$</p> <p style="padding-left: 20px;">return \perp</p> <p>$K \leftarrow_s \mathcal{K}$</p> <p>$\mathcal{L}_D := \mathcal{L}_D \cup \{(c, K)\}$</p> <p>return K</p>	<p>Oracle $H(m, c, \text{pk})$</p> <hr style="border: 0.5px solid black;"/> <p>if $\exists K$ s.t. $(m, c, \text{pk}, K) \in \mathcal{L}_H$</p> <p style="padding-left: 20px;">return K</p> <p>$K \leftarrow_s \mathcal{K}$</p> <p>if $\text{PCO}(\text{sk}^*, m, c) = 1$</p> <p style="padding-left: 20px;">if $\exists K'$ s.t. $(c, K') \in \mathcal{L}_D$</p> <p style="padding-left: 40px;">$K := K'$</p> <p style="padding-left: 20px;">else</p> <p style="padding-left: 40px;">$\mathcal{L}_D := \mathcal{L}_D \cup \{(c, K)\}$</p> <p>$\mathcal{L}_H := \mathcal{L}_H \cup \{(m, c, \text{pk}, K)\}$</p> <p>return K</p>
--	---

Fig. 5. Adversary A breaking security of the underlying OW-PCVA PKE.

As in Lemma 3, to estimate $\Pr[\text{QUERY}]$ we construct an efficient adversary A breaking OW-PCVA of PKE_1 when QUERY occurs. We define A in Fig. 5.

Notice that A perfectly simulates \mathbf{G}_1 . Indeed, the occurrence of QUERY implies that B has queried H about (m, c, pk) , in which $(m, c, \text{pk}) \in \mathcal{L}_H$ for $m = m^*$, $c = c^*$ and $\text{pk} = \text{pk}^*$. A then returns $m = m^*$. Since such event coincides with QUERY, we get that the probability of B of triggering QUERY coincides with the probability of A in winning the OW-PCVA experiment, i.e. $\Pr[\text{QUERY}] = \Pr[\text{PKE}_{\Pi_1, A}^{\text{ow-pcva}}(\lambda)] \leq \text{negl}(\lambda)$. \square

Lemma 7. $\mathbf{G}_3^B(\lambda) \equiv \text{KEM}_{\Pi_1, B, \$}^{\text{nm-cca2}^*}(\lambda)$.

Proof. Since \mathbf{G}_3 is similar to \mathbf{G}_0 , with the only difference that the encapsulation key K^* is uniform it is independent from the one obtained by querying H, for the same considerations that we did for \mathbf{G}_0 , it holds that the two distributions are identically distributed. \square

Combining the above lemmas, we get that $\text{KEM}_{\Pi_1^{\perp}, B}^{\text{nm-cca2}^*}(\lambda) \approx_c \text{KEM}_{\Pi_1^{\perp}, B, \$}^{\text{nm-cca2}^*}(\lambda)$. \square

A.3 Proof of Theorem 4

Proof. Let us assume that there exists a PPT adversary A that breaks NM-ATK* security of Π' with non-negligible probability, then we can build an efficient

distinguisher D that breaks NM-ATK* security of Π . The intuition behind the proof is that D will choose a distribution over the message space in such a way that only two messages, say K and K' , can be sampled by the the challenger playing NM-ATK* of the PKE scheme Π . In particular, D does the following:

1. Takes as input a public key pk and chooses a message distribution \mathcal{M} from which only two messages can be chosen, which we call K and K' .
2. Takes as input a challenge ciphertext c which is either an encryption of K under pk or an encryption of K' under pk .
3. Run $A(pk, c, K)$. When A asks a decapsulation-oracle query for a ciphertext \hat{c} do the following:
 - (a) Query the decryption oracle $\text{Dec}(sk^*, \cdot)$ about \hat{c} to obtain a key \hat{K} .
 - (b) Return \hat{K} to A .
4. When A returns (pk', R, c') , return (pk', R, c') as well.

Let us analyze the behavior of D . First, notice that, independently from c , D will always send the tuple (pk, c, K) to A . Recall also that the message space of c is restricted to messages K and K' . In particular, when c is an encryption of K under pk , D the view of A when run as a subroutine of D is identically distributed to its view in $\text{KEM}_{\Pi', A}^{nm-atk^*}(\lambda)$. When c is an encryption of K' under pk , since K' is uniform and independent from K , the view of A when run as a subroutine of D is distributed identically to its view in $\text{KEM}_{\Pi', A, \$}^{nm-atk^*}(\lambda)$. To summarize, the probability that D distinguishes between $\text{PKE}_{\Pi, D}^{nm-atk^*}(\lambda)$ and $\text{PKE}_{\Pi, D, \$}^{nm-atk^*}(\lambda)$ is the same of A distinguishing between $\text{KEM}_{\Pi', A, \$}^{nm-atk^*}(\lambda)$ and $\text{KEM}_{\Pi', A}^{nm-atk^*}(\lambda)$, that we assumed to be non-negligible. \square

A.4 Proof of Theorem 5

Proof. Correctness of the obtained PKE follows from the ϵ -correctness of the underlying KEM and SKE schemes. The idea behind the proof is that, given the challenge ciphertext $c^* = (c_1^*, c_2^*)$, we can use NM-ATK* security of Π^{kem} to decouple the key encapsulated in c_1^* from the key used in c_2^* to encrypt the message with the underlying SKE scheme Π^{ske} . At this point, since the encapsulated key is randomly chosen and independent from the encryption key, it is not possible to for A to distinguish between a correct encryption (i.e., where the encapsulated key and the encryption key are the same) and an encryption where the key encapsulated in c^* is randomly chosen and independent from the one used to encrypt c_2^* . This holds even if A is allowed to maul pk^* into some related public key pk' . The next step is to use the NM-ATK security of Π^{ske} to decouple m^* from the relation R , i.e. given a ciphertext c_2^* encrypting m^* , it is infeasible for an adversary to distinguish between the experiment where the relation R was checked by using either m^* or \tilde{m} . Finally, NM-ATK* security of Π^{kem} can be used to re-join together the key encapsulated in c_1^* with the key used to encrypt m in c_2^* . Let $A = A^{hy}$, the sequence of games is described in Fig. 6. The part of the proof required for a specific flavour of NM-ATK* will be highlighted with a tag [NM-ATK*].

Lemma 8. $\mathbf{G}_0(\lambda) \approx_c \mathbf{G}_1(\lambda)$.

Proof. Let us assume that A^{hy} can distinguish between \mathbf{G}_0 and \mathbf{G}_1 with non-negligible probability. We can construct an adversary A^{nm} breaking NM-ATK* security of Π^{kem} . A^{nm} behaves as follows:

1. Take as input a public key pk^* , a ciphertext c and a key \hat{K} , where either $\hat{K} = K^*$ (the key encapsulated in c) or $\hat{K} \leftarrow_s \mathcal{K}$.
2. Run $A^{hy}(\text{pk}^*)$.
 [NM-CCA*1/NM-CCA2*] When A^{hy} asks a decryption-oracle query about a ciphertext (c_1, c_2) query $\text{Decaps}(\text{sk}^*, \cdot)$ about c_1 to obtain a key K' and return $m := \text{Dec}(K', c_2)$.

Game $\mathbf{G}_0^A(\lambda)$

$(\text{pk}^*, \text{sk}^*) \leftarrow_s \text{Gen}(1^\lambda)$
 $(\mathcal{M}, s) \leftarrow_s A_1^{\text{O}1}(\text{pk})$
 $m^* \leftarrow_s \mathcal{M}$
 $(c_1^*, K^*) \leftarrow_s \text{Encaps}(\text{pk}^*)$
 $c_2^* \leftarrow_s \text{Enc}(K^*, m^*)$
 $(\text{pk}, R, c) \leftarrow_s A_2^{\text{O}2}(\mathcal{M}, \text{pk}^*, s, c^*)$
return 1 iff $\exists(m, r)$ s.t.
 $(c = \text{Enc}^{hy}(\text{pk}, m; r)) \wedge$
 $(c \neq c^* \vee \text{pk}' \neq \text{pk}) \wedge$
 $(m \neq \perp) \wedge R(m, m^*, \text{pk}, \text{pk}^*, c)$

Game $\mathbf{G}_2^A(\lambda)$

$(\text{pk}^*, \text{sk}^*) \leftarrow_s \text{Gen}(1^\lambda)$
 $(\mathcal{M}, s) \leftarrow_s A_1^{\text{O}1}(\text{pk})$
 $m^*, \tilde{m} \leftarrow_s \mathcal{M}$
 $\hat{K} \leftarrow_s \mathcal{K}$
 $(c_1^*, K^*) \leftarrow_s \text{Encaps}(\text{pk}^*)$
 $c_2^* \leftarrow_s \text{Enc}(\hat{K}, m^*)$
 $(\text{pk}, R, c) \leftarrow_s A_2^{\text{O}2}(\mathcal{M}, \text{pk}^*, s, c^*)$
return 1 iff $\exists(m, r)$ s.t.
 $(c = \text{Enc}^{hy}(\text{pk}, m; r)) \wedge$
 $(c \neq c^* \vee \text{pk}' \neq \text{pk}) \wedge$
 $(m \neq \perp) \wedge R(m, \tilde{m}, \text{pk}, \text{pk}^*, c)$

Game $\mathbf{G}_1^A(\lambda)$

$(\text{pk}^*, \text{sk}^*) \leftarrow_s \text{Gen}(1^\lambda)$
 $(\mathcal{M}, s) \leftarrow_s A_1^{\text{O}1(\cdot)}(\text{pk})$
 $m^* \leftarrow_s \mathcal{M}$
 $\hat{K} \leftarrow_s \mathcal{K}$
 $(c_1^*, K^*) \leftarrow_s \text{Encaps}(\text{pk}^*)$
 $c_2^* \leftarrow_s \text{Enc}(\hat{K}, m^*)$
 $(\text{pk}, R, c) \leftarrow_s A_2^{\text{O}2(\cdot)}(\mathcal{M}, \text{pk}^*, s, c^*)$
return 1 iff $\exists(m, r)$ s.t.
 $(c = \text{Enc}^{hy}(\text{pk}, m; r)) \wedge$
 $(c \neq c^* \vee \text{pk}' \neq \text{pk}) \wedge$
 $(m \neq \perp) \wedge R(m, m^*, \text{pk}, \text{pk}^*, c)$

Game $\mathbf{G}_3^A(\lambda)$

$(\text{pk}^*, \text{sk}^*) \leftarrow_s \text{Gen}(1^\lambda)$
 $(\mathcal{M}, s) \leftarrow_s A_1^{\text{O}1(\cdot)}(\text{pk})$
 $m^*, \tilde{m} \leftarrow_s \mathcal{M}$
 $(c_1^*, K^*) \leftarrow_s \text{Encaps}(\text{pk}^*)$
 $c_2^* \leftarrow_s \text{Enc}(K^*, m^*)$
 $(\text{pk}, R, c) \leftarrow_s A_2^{\text{O}2(\cdot)}(\mathcal{M}, \text{pk}^*, s, c^*)$
return 1 iff $\exists(m, r)$ s.t.
 $(c = \text{Enc}^{hy}(\text{pk}, m; r)) \wedge$
 $(c \neq c^* \vee \text{pk}' \neq \text{pk}) \wedge$
 $(m \neq \perp) \wedge R(m, \tilde{m}, \text{pk}, \text{pk}^*, c)$

Fig. 6. Sequence of games needed to prove Theorem 5.

When A^{hy} outputs a message distribution \mathcal{M} , take a uniform message $m \leftarrow_s \mathcal{M}$, compute $c \leftarrow_s \text{Enc}(K, m)$ and return $c^* = (c, c')$ to A^{hy} .

3. [NM-CCA2*] When A^{hy} asks a decryption-oracle query about a ciphertext (c_1, c_2) do the following:
 - if $c_1 = c$ and $c_2 = c'$ then return $m := \perp$ (i.e. the query is not admissible).
 - if $c_1 = c$ and $c_2 \neq c'$ then return $m := \text{Dec}(\hat{K}, c_2)$.
 - else, query $\text{Decaps}^{(c^*)}(\text{sk}^*, \cdot)$ about c_1 to obtain a key K' and return $m := \text{Dec}(K', c_2)$.
4. When A^{hy} outputs $(\text{pk}, R, (c_1, c_2))$, output (pk, R', c_1) , where $R'(\cdot, \cdot, \text{pk}, \text{pk}^*, c) = R(\cdot, \cdot, \text{pk}, \text{pk}^*, c)$.

Notice that, since the only difference between \mathbf{G}_0 and \mathbf{G}_1 is that in \mathbf{G}_1 the key is chosen at random, the only hope for the adversary A^{hyb} to distinguish between the two hybrids is by finding a relation holding between pk , pk^* , and c that is satisfied in \mathbf{G}_0 but not in \mathbf{G}_1 (or vice-versa). Hence, $R'(\cdot, \cdot, \text{pk}, \text{pk}^*, c) = R(\cdot, \cdot, \text{pk}, \text{pk}^*, c)$ is indeed a suitable relation for A^{nm} . When \hat{K} taken as input by A^{nm} is K^* , then A^{nm} perfectly simulates \mathbf{G}_0 . When \hat{K} taken as input by A^{nm} is randomly chosen, A^{nm} perfectly simulates \mathbf{G}_1 . If A^{hy} distinguishes between \mathbf{G}_0 and \mathbf{G}_1 with non-negligible probability, then A^{nm} breaks NM-ATK* security of the underlying KEM scheme with non-negligible probability. This leads to a contradiction. \square

Lemma 9. $\mathbf{G}_1(\lambda) \approx_c \mathbf{G}_2(\lambda)$.

Proof. Let us assume that A^{hy} can distinguish between \mathbf{G}_1 and \mathbf{G}_2 with non-negligible probability, we construct an adversary A^{atk} breaking NM-ATK security of Π^{ske} . A^{atk} behaves as follows:

1. Receive as input a key K .
2. Generate a pair $(\text{pk}^*, \text{sk}^*) \leftarrow_s \text{Gen}^{kem}(1^\lambda)$.
3. Run $A^{hy}(\text{pk}^*)$.
 - [NM-CCA*1/NM-CCA2*] When A^{hy} asks a decryption-oracle query about a ciphertext (c_1, c_2) , query the decryption oracle $\text{Dec}(\hat{K}, \cdot)$ of the NM-CCA experiment about c_2 to obtain a message m .
4. When A^{hy} outputs a message distribution \mathcal{M} , output \mathcal{M} to the challenger.
5. When receiving a ciphertext c' from the challenger, compute $(c_1^*, K^*) \leftarrow_s \text{Encaps}(\text{pk}^*)$, and output (c_1^*, c') to A^{hy} .
6. [NM-CCA2*] When A^{hy} asks a decryption-oracle query about a ciphertext (c_1, c_2) do the following:
 - if $c_1 = c_1^*$ and $c_2 = c'$ then return $m := \perp$ (i.e. the query is not admissible).
 - else, query $\text{Dec}^{(c')}(K, \cdot)$ about c_2 to obtain $m := \text{Dec}(K', c_2)$. Then, outputs m to A^{hyb} .
7. When A^{hy} outputs $(\text{pk}, R, (c_1, c_2))$, A^{atk} output (R', c_2) to the challenger, where $R'(m_0, m_1) = R(m_0, m_1, \cdot, \cdot, \cdot)$. The challenger either checks $R'(m, m^*) = 1$ where $m^* := \text{Dec}(\hat{K}, c')$ if A^{atk} is in $\text{SKE}_{\Pi^{ske}, A^{atk}}^{nm-atk}$ or checks if $R'(m, \tilde{m}) = 1$ if \tilde{m} is an randomly chosen message independent from c^* . Note that the

only difference between \mathbf{G}_1 and \mathbf{G}_2 is that the game checks that m^* given as an input to R is encrypted in c_2^* , whereas in \mathbf{G}_2 the relation R takes as an input a random \tilde{m} . Thus, the only hope for A^{hyb} in distinguishing between the two hybrids is by finding a relation holding between m and m^* but not between m and \tilde{m} (or vice-versa). Thus, we are allowed to cast $R'(m_0, m_1)$ as $R(m_0, m_1, \cdot, \cdot, \cdot)$. When the relation R takes m^* in input, then A^{atk} perfectly simulates \mathbf{G}_1 . When the relation R takes a random \tilde{m} in input, A^{atk} perfectly simulates \mathbf{G}_2 .

If A^{hy} distinguishes between \mathbf{G}_1 and \mathbf{G}_2 with non-negligible probability, then A^{atk} breaks NM-ATK security of the underlying SKE scheme with non-negligible probability. This leads to a contradiction. \square

Lemma 10. $\mathbf{G}_2^A(\lambda) \approx_c \mathbf{G}_3^A(\lambda)$

Proof. Let us assume that A^{hy} can distinguish between \mathbf{G}_2 and \mathbf{G}_3 with non-negligible probability, we construct an adversary A^{nm} breaking NM-ATK* security of Π^{kem} . A^{nm} behaves as follows:

1. Takes as input a public key pk^* , a ciphertext c and a key \hat{K} , where either $\hat{K} = K^*$ (the key encapsulated in c) or $\hat{K} \leftarrow_{\$} K$.
2. Run $A^{hy}(\text{pk}^*)$.
 [NM-CCA*1/NM-CCA2*] When A^{hy} asks a decryption-oracle query about a ciphertext (c_1, c_2) query $\text{Decaps}(\text{sk}^*, \cdot)$ about c_1 to obtain a key K' and return $m := \text{Dec}(K', c_2)$.
3. When A^{hy} outputs a message distribution \mathcal{M} , take two uniform messages $m, \tilde{m} \leftarrow_{\$} \mathcal{M}$, compute $c \leftarrow_{\$} \text{Enc}(K, m)$ and return $c^* = (c, c')$ to A^{hy} .
4. [NM-CCA2*] When A^{hy} asks a decryption-oracle query about a ciphertext (c_1, c_2) do the following:
 - if $c_1 = c$ and $c_2 = c'$ then return $m := \perp$ (i.e. the query is not admissible).
 - if $c_1 = c$ and $c_2 \neq c'$ then return $m := \text{Dec}(\hat{K}, c_2)$.
 - else, query $\text{Decaps}^{(c^*)}(\text{sk}^*, \cdot)$ about c_1 to obtain a key K' and return $m := \text{Dec}(K', c_2)$.
5. When A^{hy} outputs $(\text{pk}, R, (c_1, c_2))$, output (pk, R', c_1) , where $R'(\cdot, \cdot, \text{pk}, \text{pk}^*, c) = R(\cdot, \cdot, \text{pk}, \text{pk}^*, c)$.

Notice that, since the only difference between \mathbf{G}_2 and \mathbf{G}_3 is that in \mathbf{G}_2 the key is chosen at random, the only hope for the adversary A^{hyb} to distinguish between the two hybrids is by finding a relation holding between pk , pk^* , and c that is satisfied in \mathbf{G}_2 but not in \mathbf{G}_3 (or vice-versa). Hence, R' is a suitable relation for A^{nm} . When the key \hat{K} taken as input by A^{nm} is randomly chosen, it perfectly simulates \mathbf{G}_2 . When the key \hat{K} taken as input by A^{nm} is K^* , then A^{nm} perfectly simulates \mathbf{G}_3 . If A^{hy} distinguishes between \mathbf{G}_2 and \mathbf{G}_3 with non-negligible probability, then A^{nm} breaks NM-ATK* security of the underlying KEM scheme with non-negligible probability. This leads to a contradiction. \square

It is easy to see that $\mathbf{G}_0^A(\lambda) \equiv \text{PKE}_{\Pi^{hy}, A}^{nm-atk*}(\lambda)$ and that $\mathbf{G}_3^A(\lambda) \equiv \text{PKE}_{\Pi^{hy}, A, \$}^{nm-atk*}(\lambda)$. by combining the above lemmas, we have that $\text{PKE}_{\Pi^{hy}, A}^{nm-atk*}(\lambda) \approx_c \text{PKE}_{\Pi^{hy}, A, \$}^{nm-atk*}(\lambda)$. \square

References

1. Barbosa, M., Farshim, P.: Relations among notions of complete non-malleability: Indistinguishability characterisation and efficient construction without random oracles. In: Steinfeld, R., Hawkes, P. (eds.) ACISP (2010)
2. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, 19–22 October 1997, pp. 394–403. IEEE Computer Society (1997)
3. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 26–45. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0055718>
4. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_41
5. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. *J. Cryptol.* **21**(4), 469–491 (2008)
6. Bellare, M., Sahai, A.: Non-malleable encryption: equivalence between two notions, and an indistinguishability-based characterization. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 519–536. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_33
7. Boldyreva, A., Fischlin, M.: Analysis of random oracle instantiation scenarios for OAEP and other practical schemes. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 412–429. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_25
8. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0055717>
9. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography (extended abstract). In: Koutsougeras, C., Vitter, J.S. (eds.) Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, 5–8 May 1991, New Orleans, Louisiana, USA, pp. 542–552. ACM (1991)
10. Duman, J., Hövelmanns, K., Kiltz, E., Lyubashevsky, V., Seiler, G.: Faster lattice-based kems via a generic Fujisaki-Okamoto transform using prefix hashing. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, 15–19 November 2021, pp. 2722–2737. ACM (2021)
11. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the fiat-Shamir transform. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34931-7_5
12. Fischlin, M.: Completely non-malleable schemes. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 779–790. Springer, Heidelberg (2005). https://doi.org/10.1007/11523468_63
13. Friolo, D., Salvino, M., Venturi, D.: On the complete non-malleability of the Fujisaki-Okamoto transform. *Cryptology ePrint Archive*, Paper 2022/1654 (2022). <https://eprint.iacr.org/2022/1654>

14. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_34
15. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10677, pp. 341–371. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_12
16. Libert, B., Yung, M.: Efficient completely non-malleable public key encryption. In: Abramsky, S., Gavoiile, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 127–139. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14165-2_12
17. Nagao, W., Manabe, Y., Okamoto, T.: On the equivalence of several security notions of key encapsulation mechanism. IACR Cryptology ePrint Archive, p. 268 (2006). <https://eprint.iacr.org/2006/268>
18. NIST: Nist announces first four quantum-resistant cryptographic algorithms (2022). <https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms>
19. Pass, R., Shelat, A., Vaikuntanathan, V.: Relations among notions of non-malleability for encryption. In: Kurosawa, K. (ed.) ASIACRYPT, vol. 4833, pp. 519–535 (2007)
20. Rackoff, C., Simon, D.R.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 433–444. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_35
21. Sepahi, R., Steinfeld, R., Pieprzyk, J.: Lattice-based completely non-malleable PKE in the standard model (poster). In: Parampalli, U., Hawkes, P. (eds.) ACISP 2011. LNCS, vol. 6812, pp. 407–411. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22497-3_26
22. Sepahi, R., Steinfeld, R., Pieprzyk, J.: Lattice-based completely non-malleable public-key encryption in the standard model. *Des. Codes Cryptogr.* **71**(2), 293–313 (2014)
23. Ventre, C., Visconti, I.: Completely non-malleable encryption revisited. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 65–84. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78440-1_5



Optimal Security Notion for Decentralized Multi-Client Functional Encryption

Ky Nguyen¹ , Duong Hieu Phan² , and David Pointcheval¹ 

¹ DIENS, École normale supérieure, CNRS, Inria, PSL University, Paris, France
`{ky.nguyen,david.pointcheval}@ens.fr`

² LTCI, Telecom Paris, Institut Polytechnique de Paris, Palaiseau, France
`hieu.phan@telecom-paris.fr`

Abstract. Research on (Decentralized) Multi-Client Functional Encryption (or (D)MCFE) is very active, with interesting constructions, especially for the class of inner products. However, the security notions have been evolving over the time. While the target of the adversary in distinguishing ciphertexts is clear, legitimate scenarios that do not consist of trivial attacks on the functionality are less obvious. In this paper, we wonder whether only trivial attacks are excluded from previous security games. And, unfortunately, this was not the case.

We then propose a stronger security notion, with a large definition of admissible attacks, and prove it is optimal: any extension of the set of admissible attacks is actually a trivial attack on the functionality, and not against the specific scheme. In addition, we show that all the previous constructions are insecure w.r.t. this new security notion. Eventually, we propose new DMCFE schemes for the class of inner products that provide the new features and achieve this stronger security notion.

Keywords: Functional Encryption · Corruptions · Security Notions

1 Introduction

Decentralized Multi-Client Functional Encryption. Multi-Input Functional Encryption (MIFE) and Multi-Client Functional Encryption (MCFE), together with their decentralized variants [15, 21, 22], have been receiving a strong interest from the cryptographic community. They generalize the nice functional encryption primitive [13, 29] where the single input x , in the encryption procedure, is split into an input vector (x_1, \dots, x_n) , and the components can be encrypted independently, possibly by different senders/clients in MCFE. An index i for each component, and a (typically time-based) tag tag for MCFE, are used for every encryption $c_i = \text{Enc}(i, \text{tag}, x_i)$. From the n encrypted components under the same tag tag , anyone owning a functional decryption key dk_f , for the n -ary function f , can compute $f(x_1, \dots, x_n)$ but nothing else about the individual x_i 's. In this paper, we focus on a standard and optimal security model for the most general form of MCFE, namely *decentralized* MCFE, where the generation of functional decryption keys is also split between multiple clients.

Previous Corruption Model for (D)MCFE. In previous (D)MCFE, encryption was claimed to require a private key ek_i per client, for each component c_i , because of deterministic encryption. Then, some of these keys might get corrupted. In DMCFE, where multiple clients contribute to generate the decryption functional keys and also own secret keys sk_i , and some can get corrupted. Therefore, there exists potential corruption of two categories of keys regarding DMCFE that need to be dealt with: a private encryption key ek_i for encryption and a secret key sk_i for generating functional keys. The proposed corruption model in the work on DMCFE by Chotard *et al.* [15] is: when an adversary corrupts a client i , it receives both $(\text{sk}_i, \text{ek}_i)$. However, this does not reflect the real-life situation. In fact, the encryption keys ek_i 's and the secret keys sk_i 's can have different levels of protection (sk_i looks more critical than ek_i) and can be stored on different devices. This is thus a strong restriction to get both keys in case of corruption. Actually, this corruption model was employed in the previous DMCFE constructions for inner products $f_{\mathbf{y}}(\mathbf{x}) = \langle \mathbf{x}, \mathbf{y} \rangle$, as the numbers of sk_i 's and ek_i 's are equal, and in most of them particularly, sk_i is either included in ek_i , *e.g.* [15, 16, 24], or they are the same, *e.g.* [2, 3, 7]¹. But this might not always be the case. Specifically, for quadratic functions computing $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{x}^\top \mathbf{A} \mathbf{x}$ as considered in [6, 8], one could have n^2 secret keys sk_j for the square matrix \mathbf{A} and n encryption keys ek_i only for the input vector \mathbf{x} . Hence, the holders of sk_j 's and ek_i 's might differ.

Previous Notions of Admissible Attacks against (D)MCFE. Generally, studying an advanced cryptographic primitive involves formalizing the ubiquitous perception of trivial attacks when devising its security notion, those that exploit only the *functionality* of the primitive to trivially break any specific constructions. A standard example is the case of *identity-based encryption* [11, 12, 18, 30], of which the widely agreed security notion forbids the adversary to obtain the secret key of any identity that could decrypt the challenge ciphertext. In our case of (D)MCFE, everything becomes much more complicated due to the computational aspect of the function class and the corruption in multi-user setting. Following the introduction of (D)MCFE in the seminal paper [15], to the best of our knowledge, all follow-up studies on (D)MCFE, for instance [2, 3, 7, 16, 17, 24], administered an *admissibility condition* in order to prohibit trivial attacks, and restricted particularly adversaries to asking the challenge components $\mathbf{x}_0^*[i] = \mathbf{x}_1^*[i]$ in case of a corrupted i . Attacks that satisfy the admissibility condition are called *admissible attacks*. Nonetheless, there was no satisfactory justification for such a restriction, except that all the constructions used a deterministic encryption, and so the corruption of ek_i could allow to re-encrypt $\mathbf{x}_0^*[i]$ and compare with the challenge ciphertext. This was thus also the similar argument to support private encryption keys. Since then, relaxing the foregoing constraint was widely believed to be insurmountable for constructing (D)MCFE schemes and proving their security.

¹ The work of [7] constructs *function-hiding dynamic decentralized FE*, which directly yields a DMCFE with a stronger property of function secrecy. Even though their proposed security model captures separated corruption of ek_i and sk_i , implying they are different, their dynamic decentralized FE construction uses the same key for both and so does the resulting DMCFE, *i.e.* $\text{sk}_i = \text{ek}_i$ for every i .

An Improved Security Model for DMCFE. Since previous security notions of DMCFE turn out unstable, the main goal of this paper is to propose a fair and optimal security model.

SEPARATING CORRUPTIONS OF ek_i AND sk_i . Our first step is thus to *separate* the corruption of sk_j from that of ek_i , *i.e.* the adversary must specify which type of keys it wants to corrupt. This gives more flexibility to the adversary. However, its goal remains the same: distinguish between the encryption of \mathbf{x}_0^* and \mathbf{x}_1^* in the challenge ciphertext. We notice that this new corruption model captures the previous “both-or-nothing” model in previous works and any scheme that is secure in this new fine-grained model will also be secure in the old one. A very recent work by Agrawal *et al.* [7] also defined a security model with similar fine-grained corruption, though as mentioned earlier (see footnote 3) their subsequent DMCFE scheme for inner products has $sk_i = ek_i$ for every i and by corrupting one an adversary will obtain both keys.

REFINING ADMISSIBILITY FOR A STRONGER SECURITY. Our next objective consists in challenging the belief from previous admissibility conditions and relaxing the restriction $\mathbf{x}_0^*[i] = \mathbf{x}_1^*[i]$ in case of a corrupted i . A more relaxed admissibility means more attacks will be considered, leading to a stronger security notion. To summarize, we revise the security model for (D)MCFE and

1. We provide a new security model for DMCFE under separated corruption of keys and less restrictive admissibility condition. Our security model covers the security model in all previous works, in the sense that being secure in the former implies being secure in the latter.

In Sect. 3.1, we give the intuition of our new formulation for admissibility condition. This new definition will require probabilistic encryption, which exclude the need of private encryption keys. Our security model will thus also consider public-key encryption, as some security still holds when all the encryption keys ek_i are corrupted. Note however this might make sense for limited classes of functions only and becomes completely meaningless when both (ek_i, sk_i) can be corrupted at once.

Optimality. At the core of our new security model is a more relaxed admissibility condition. Up to this point one may well wonder if there is still room to relax our condition, in the same way we have done to the admissibility condition put forth since the birth of DMCFE in [15]. Our goal is to analyze this question in a rigorous manner. This turns out to be notoriously hard because we aim to settle this infamous problem with satisfactory justifications whenever a new condition is introduced. Intuitively, since all prior works did not elaborate formally whether an admissibility condition must be respected or it is just optional, we have to start from scratch to formalize how “indispensable” a condition is. We thus address this *optimality* question and this leads to our second contribution:

2. We provide a new framework to prove the optimality of our new notion of *admissible attacks*. More formally, this allows us to show that any non-admissible attack would actually break any efficient construction for the

functionality. This proves that we only exclude attacks that are at the functionality level and not at the scheme level.

We believe that the conceptual message from our methodology is one main contribution of this paper. We refer to Sect. 3.3 for a detailed explanation of our modeling choices as well as the encountered problems.

Impact and Feasibility. While we have shown our security notion to be optimal w.r.t. the functionality for a class of functions, there are two remaining questions, with respect to this new admissibility notion: are the previous constructions secure? Can we construct concrete schemes for non-trivial functionalities?

First, we can show that the class of inner products is a non-trivial class. Furthermore, it has been widely studied, with several candidates: the DDH-based MCFE for inner products from [2, 15, 17] cannot be proven secure in our model, due to the following attack, which was artificially excluded in the previous security models. For any corrupted key ek_i , it was required that $\mathbf{x}_0^*[i] = \mathbf{x}_1^*[i]$, because of the deterministic encryption: an adversary corrupts client 1 among n clients to get ek_1 , then queries the function \mathbf{y} with $\mathbf{y}[1] = 0$ and challenges $(\mathbf{x}_0^*[i], \mathbf{x}_1^*[i])_{i \in [n]}$ such that $\mathbf{x}_0^*[1] \neq \mathbf{x}_1^*[1]$ and $\langle \mathbf{x}_0^*, \mathbf{y} \rangle = \langle \mathbf{x}_1^*, \mathbf{y} \rangle$. Then, the adversary encrypts $\mathbf{x}_1^*[1]$ on their own using ek_1 . By comparing with the obtained ciphertext on $\mathbf{x}_b^*[1]$, such an adversary can decide correctly on b . In addition to these DDH-based constructions, in a work by Libert and Titu [24], the authors proposed the first LWE-based MCFE in the standard model. The ciphertext components of this scheme is somewhat randomized by some small Gaussian error, but the above attack still works by choosing $\mathbf{x}_0^*[1] \neq \mathbf{x}_1^*[1]$ that are far from each other, then deciding based on the norm of the two ciphertexts' difference². We note that the above attack gives a byproduct that complements our first contribution

1-bis. Our security model is strictly stronger than the security model in almost all previous works, in the sense that prior concrete schemes cannot be proven secure in ours.

Besides the theoretical part introducing and proving our optimal security notion for DMCFE, we also propose new constructions in the DDH setting which meet the proposed level of security. This requires a number of new technical ideas, in particular a technique for achieving admissibility *via revocation* (in a different way than [5]) and using *dual pairing vector spaces* (DPVS) [26–28], to build a probabilistic encryption scheme.

Roughly speaking, our new admissibility when translated for the particular cases of inner-products introduces one condition that for all corrupted clients i , for ek_i , for all functional key query \mathbf{y} , it must hold that

$$(\mathbf{x}_0^*[i] - \mathbf{x}_1^*[i]) \cdot \mathbf{y}[i] = 0 \quad . \quad (1)$$

Previous security models required $\mathbf{x}_0^*[i] = \mathbf{x}_1^*[i]$, but we now have to deal with the case $\mathbf{x}_0^*[i] \neq \mathbf{x}_1^*[i]$ and $\mathbf{y}[i] = 0$ additionally. A necessary condition is that our

² We use the metrics employed in the context of the LWE-based (D)MCFE in [24].

encryption must be probabilistic (otherwise, the attack described in the previous paragraph applies). However, that is *not* enough because we want semantic security for the ciphertext component ct_i of $\mathbf{x}_b^*[i]$ as well, where $b \stackrel{\$}{\leftarrow} \{0, 1\}$ is the challenge bit. When we view this problem under the lens of revocation systems, similarities emerge: as soon as the special value 0 is set for $\mathbf{y}[i]$, we want to nullify the ability for recovering information about $\mathbf{x}_b^*[i]$. The foregoing fits well in the context of revocation. Conveniently, the work by Agrawal *et al.* [5] solved the “dual” problem, namely using IPFE to construct revocation systems, and along the way, the authors of [5] presented a DDH-based IPFE that we can embed locally into the vectors in DPVS, components by components. We leverage this idea to concoct DPVS-based DMCFE schemes for inner-product functionality and achieve security under the condition (1). In the end, our third contribution is

3. We demonstrate the feasibility of our new security model by presenting DDH-based DMCFE schemes for inner products over polynomially bounded ranges using pairings, the first concrete scheme whose security holds against fine-grained corruptions and a less restrictive admissibility.

More high-level details are provided in Sect. 3.4.

2 Preliminaries

We write $[n]$ to denote the set $\{1, 2, \dots, n\}$ for an integer n . For any $q \geq 2$, we let \mathbb{Z}_q denote the ring of integers with addition and multiplication modulo q . For a prime q and an integer N , we denote by $GL_N(\mathbb{Z}_q)$ the general linear group of degree N over \mathbb{Z}_q . We write vectors as row-vectors, unless stated otherwise. For a vector \mathbf{x} of dimension n , the notation $\mathbf{x}[i]$ indicates the i -th coordinate of \mathbf{x} , for $i \in [n]$. We will follow the implicit notation in [20] and use $\llbracket a \rrbracket$ to denote g^a in a cyclic group \mathbb{G} of prime order q generated by g , given $a \in \mathbb{Z}_q$. This implicit notation extends to matrices and vectors having entries in \mathbb{Z}_q . We use the shorthand *ppt* for “probabilistic polynomial time”. In the security proofs, whenever we use an ordered sequence of games $(\mathbb{G}_0, \mathbb{G}_1, \dots, \mathbb{G}_i, \dots, \mathbb{G}_L)$ indexed by $i \in \{0, 1, \dots, L\}$, we refer to the predecessor of \mathbb{G}_j by \mathbb{G}_{j-1} , for $j \in [L]$.

2.1 Hardness Assumptions

We state the assumptions needed for our constructions.

Definition 1. *In a cyclic group \mathbb{G} of prime order q , the **Decisional Diffie-Hellman** (DDH) problem is to distinguish the distributions*

$$D_0 = \{(\llbracket 1 \rrbracket, \llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket ab \rrbracket)\} \quad D_1 = \{(\llbracket 1 \rrbracket, \llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)\}.$$

for $a, b, c \stackrel{\$}{\leftarrow} \mathbb{Z}_q$. The DDH assumption in \mathbb{G} assumes that no *ppt* adversary can decide the DDH problem with non-negligible probability.

In the bilinear setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, g_t, \mathbf{e}, q)$, the **Symmetric eXternal Diffie-Hellman (SXDH)** assumption makes the DDH assumption in both \mathbb{G}_1 and \mathbb{G}_2 .

Definition 2. *In the bilinear setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, g_t, \mathbf{e}, q)$, the **Decisional Bilinear Diffie-Hellman (DBDH)** problem is to distinguish the distributions*

$$D_0 = \{([\![a]\!]_1, [\![b]\!]_1, [\![b]\!]_2, [\![c]\!]_2, [\![abc]\!]_t)\} \quad D_1 = \{([\![a]\!]_1, [\![b]\!]_1, [\![b]\!]_2, [\![c]\!]_2, [\![r]\!]_t)\}.$$

for $a, b, c, r \xleftarrow{\$} \mathbb{Z}_q$. The DBDH assumption in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, g_t, \mathbf{e}, q)$ assumes that no ppt adversary can decide the DBDH problem with non-negligible probability.

2.2 Dual Pairing Vector Spaces

Our constructions rely on the *Dual Pairing Vector Spaces (DPVS)* framework in prime-order bilinear group setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, g_t, \mathbf{e}, q)$ and $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ are all written additively. The DPVS technique dates back to the seminal work by Okamoto-Takashima [26–28] aiming at adaptive security for ABE as well as IBE, together with the *dual system methodology* introduced by Waters [31]. In [23], the setting for dual systems is composite-order bilinear groups. Continuing on this line of works, Chen *et al.* [14] used prime-order bilinear groups under the SXDH assumption. Let us fix $N \in \mathbb{N}$ and consider \mathbb{G}_1^N having N copies of \mathbb{G}_1 . Any $\mathbf{x} = [\![x_1, \dots, x_N]\!]_1 \in \mathbb{G}_1^N$ is identified as the vector $(x_1, \dots, x_N) \in \mathbb{Z}_q^N$. There is no ambiguity because \mathbb{G}_1 is a cyclic group of prime order q . The $\mathbf{0}$ -vector is $\mathbf{0} = [\!(0, \dots, 0)\!]_1$. The addition of two vectors in \mathbb{G}_1^N is defined by coordinate-wise addition. The scalar multiplication of a vector is defined by $t \cdot \mathbf{x} := [\![t \cdot (x_1, \dots, x_N)]\!]_1$, where $t \in \mathbb{Z}_q$ and $\mathbf{x} = [\!(x_1, \dots, x_N)\!]_1$. The additive inverse of $\mathbf{x} \in \mathbb{G}_1^N$ is defined to be $-\mathbf{x} := [\!(-x_1, \dots, -x_N)\!]_1$. Viewing \mathbb{Z}_q^N as a vector space of dimension N over \mathbb{Z}_q with the notions of bases, we can obtain naturally a similar notion of bases for \mathbb{G}_1^N . More specifically, any invertible matrix $B \in GL_N(\mathbb{Z}_q)$ identifies a basis \mathbf{B} of \mathbb{G}_1^N , whose i -th row \mathbf{b}_i is $[\![B^{(i)}]\!]_1$, where $B^{(i)}$ is the i -th row of B . The canonical basis \mathbf{A} of \mathbb{G}_1^N consists of $\mathbf{a}_1 := [\!(1, 0, \dots, 0)\!]_1, \mathbf{a}_2 := [\!(0, 1, 0, \dots, 0)\!]_1, \dots, \mathbf{a}_N := [\!(0, \dots, 0, 1)\!]_1$. It is straightforward that we can write $\mathbf{B} = B \cdot \mathbf{A}$ for any basis \mathbf{B} of \mathbb{G}_1^N corresponding to an invertible matrix $B \in GL_N(\mathbb{Z}_q)$, for the change of basis. We write $\mathbf{x} = (x_1, \dots, x_N)_{\mathbf{B}}$ to indicate the representation of \mathbf{x} in the basis \mathbf{B} , i.e. $\mathbf{x} = \sum_{i=1}^N x_i \cdot \mathbf{b}_i$. Given a basis $\mathbf{B} = (\mathbf{b}_i)_{i \in [N]}$ of \mathbb{G}_1^N , we define \mathbf{B}^* to be a basis of \mathbb{G}_2^N by first defining $B' := (B^{-1})^\top$ and the i -th row \mathbf{b}_i^* of \mathbf{B}^* is $[\![B'^{(i)}]\!]_2$. It holds that $B \cdot (B')^\top = I_N$ the identity matrix and $\mathbf{b}_i \times \mathbf{b}_j^* = [\![\delta_{i,j}]\!]_t$ for every $i, j \in [N]$, where $\delta_{i,j} = 1$ if and only if $i = j$. Treating \mathbb{G}_2^N similarly, we can furthermore define a product of two vectors $\mathbf{x} = [\!(x_1, \dots, x_N)\!]_1 \in \mathbb{G}_1^N, \mathbf{y} = [\!(y_1, \dots, y_N)\!]_2 \in \mathbb{G}_2^N$ by $\mathbf{x} \times \mathbf{y} := \prod_{i=1}^N \mathbf{e}(\mathbf{x}[i], \mathbf{y}[i]) = [\![(x_1, \dots, x_N), (y_1, \dots, y_N)]\!]_t$. We call the pair $(\mathbf{B}, \mathbf{B}^*)$ *dual orthogonal bases* of $(\mathbb{G}_1^N, \mathbb{G}_2^N)$. If \mathbf{B} is constructed by a random invertible matrix $B \xleftarrow{\$} GL_N(\mathbb{Z}_q)$, we call the resulting $(\mathbf{B}, \mathbf{B}^*)$ a pair of random

dual bases. A DPVS is a bilinear group setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, g_t, \mathbf{e}, q, N)$ with dual orthogonal bases. In this work, we also use extensively *basis changes* over dual orthogonal bases, the details are recalled in the full version [25].

3 Technical Overview

3.1 Motivations for a Refinement on Admissibility

In the seminal work on DMCFE for a function class \mathcal{F} by Chotard *et al.* [15], the authors define the security game with oracles

(Initialize, Corrupt, LoR, DKeyGenShare, Enc, Finalize)

between a challenger and an adversary. The oracle **Initialize** sets up the parameters, including the number of clients n and their secret-encryption key pairs $(\text{sk}_i, \text{ek}_i)$. The oracle **DKeyGenShare** produces functional key components for $F \in \mathcal{F}$ using sk_i under some function tag $\text{tag-f} \in \text{Tag}$, while **LoR** is the left-or-right oracle, which outputs the challenge ciphertext component of $\mathbf{x}_b^*[i]$ under $\text{tag} \in \text{Tag}$ upon receiving $(\mathbf{x}_0^*[i], \mathbf{x}_1^*[i])$ for $b \xleftarrow{\$} \{0, 1\}$. An adversary can corrupt any client i of his choice by querying **Corrupt** so as to receive *both* $(\text{sk}_i, \text{ek}_i)$.

In the end, a set of conditions is specified such that the adversary wins the game only when they conform to these conditions and outputs a correct b' equal to the challenge bit b . The main reason there are such conditions is to focus only on the scenarios where a notion of semantic security really makes sense in this DMCFE setting. In this paper we call an attack that satisfies such conditions an *admissible* attack. Checking these conditions is done by a **Finalize** procedure in the security game, according to the sets \mathcal{C} of corrupted clients (asked to **Corrupt**), \mathcal{H} of honest clients, and \mathcal{Q} of key queries (asked to **DKeyGenShare**) during the attack. To recall from [15, Definition 2, 5], the adversary's output is ignored and replaced by a random bit, *i.e.* the attack is NOT admissible, if one of the following holds:

1. There exists a corrupted client $i \in \mathcal{C}$ such that the i -th components of the challenge messages $(\mathbf{x}_0^*, \mathbf{x}_1^*)$ are not the same, *i.e.* $\mathbf{x}_0^*[i] \neq \mathbf{x}_1^*[i]$.
2. There exists a tag $\text{tag} \in \text{Tag}$ (respectively, $\text{tag-f} \in \text{Tag}$) and $i \neq j \in \mathcal{H}$ such that the j -th challenge component (respectively, key component) is queried but the i -th challenge component (respectively, key component) is not.
3. None of the two above conditions are satisfied, but there exists a function F queried to **DKeyGenShare** that differs on $(\mathbf{x}_0^*, \mathbf{x}_1^*)$, *i.e.* $F(\mathbf{x}_0^*) \neq F(\mathbf{x}_1^*)$.

Our observation is that only the condition 3 can be justified for the sake of avoiding trivial attacks, while the other conditions 1 and 2 do not have satisfactory explanations. About condition 1, we have seen from the attacks in the paragraph **Impact and Feasibility** of Sect. 1 that this condition is artificial and unfortunately excludes also non-trivial attacks. About condition 2, follow-up works [16, 17] and other results on the subject [2–4, 19] show that we

can completely remove this constraint. Our objective now becomes devising a less restrictive admissible condition, which should capture and generalize only condition 3. We recall that a less restrictive condition implies *more* attacks will be considered non-trivial and we obtain a *stronger* security model.

3.2 Towards a New Admissibility Condition Under Separated Corruption of Keys

Our first step is to *separate* the corruption of sk_i from that of ek_i , *i.e.* the adversary has to specify which type of keys with respect to component i he wants to corrupt. All prior works allow the adversary to corrupt both keys *at once*. This separation helps us define in a finer way which information the adversary can deduce using each type of corrupted keys, and thus even deal with public-key encryption. As a result, we have sets of corrupted and honest clients $(\mathcal{C}_{\text{skey}}, \mathcal{H}_{\text{skey}}), (\mathcal{C}_{\text{ekey}}, \mathcal{H}_{\text{ekey}})$, independently for the secret keys $(\text{sk}_j)_j$ and the encryption keys $(\text{ek}_i)_i$. This even allows to have independent sets of clients owning the secret keys $(\text{sk}_j)_j$ and the encryption keys $(\text{ek}_i)_i$. Our complete security experiment can be found in Fig. 1. Being already mentioned in **Previous Corruption Model for (D)MCFE** of Sect. 1, to the best of our knowledge, almost all prior proposed constructions of (D)MCFE can not handle separate corruption of ek_i and sk_i , for example, see [2, 3, 7, 15, 16, 24], despite the fact that a such separation is meaningful and is indeed discussed notably in the security model of [7].

Next, we represent an n -ary function $F : \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \rightarrow \mathcal{R}$ of a function class \mathcal{F} by a length- m vector of parameters from $\text{Param}_1 \times \cdots \times \text{Param}_m$, given by a deterministic encoding $\mathbf{p} : \mathcal{F} \rightarrow \text{Param}_1 \times \cdots \times \text{Param}_m$ and m can be different from n . Given such representations as vectors for both inputs and functions, we define the notion *deducible inputs and functions* (see Definition 3). More specifically, let $\mathcal{H}_{\text{inp}} \subseteq [n], \mathcal{H}_{\text{func}} \subseteq [m]$ and suppose we are given $\mathbf{x}_{\text{inp}} \in (\mathcal{D}_i)_{i \in \mathcal{H}_{\text{inp}}}$ and $\mathbf{y}_{\text{func}} \in (\text{Param}_i)_{i \in \mathcal{H}_{\text{func}}}$ as lists of inputs and parameters that are indexed by $\mathcal{H}_{\text{inp}}, \mathcal{H}_{\text{func}}$ respectively. A vector \mathbf{z} is *deducible* from \mathbf{x}_{inp} if their coordinates at positions in \mathcal{H}_{inp} are the same. Similarly, a function G is *deducible* from \mathbf{y}_{func} if its parameters coincide at positions in $\mathcal{H}_{\text{func}}$ with \mathbf{y}_{func} . Intuitively, the lists $(\mathbf{x}_{\text{inp}}, \mathbf{y}_{\text{func}})$ play the role of “honest” predetermined input’s components and function’s parameters, whilst the deducible (\mathbf{z}, G) signifies what the adversary can infer by manipulating the remaining “corrupted” parts of the input and function.

Being equipped with this notion of deducible inputs and functions, our admissible condition is formulated as:

Given the sets $(\mathcal{H}_{\text{skey}}, \mathcal{H}_{\text{ekey}})$, an *attack* is NOT admissible if there exist $\text{tag}, \text{tag-f} \in \text{Tag}$, a function $F \in \mathcal{F}$ with parameters $\mathbf{y} = (y_j)_{j \in [m]}$, two challenges $(\mathbf{x}_0^*, \mathbf{x}_1^*) := (x_i^{(0)}, x_i^{(1)})_{i \in [n]}$ such that $(F, \text{tag-f})$ is queried to **DKeyGenShare**, $((x_i^{(0)}, x_i^{(1)})_{i \in [n]}, \text{tag})$ is queried to **LoR** and there exists a pair $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)})$ deducible from $(\mathbf{x}_{\text{ekey}}^{(0)}, \mathbf{x}_{\text{ekey}}^{(1)})$, a function G deducible from \mathbf{y}_{skey} satisfying $G(\mathbf{z}^{(0)}) \neq G(\mathbf{z}^{(1)})$ where we define $\mathbf{y}_{\text{skey}} := (y_i)_{i \in \mathcal{H}_{\text{skey}}}$ and for $b \in \{0, 1\}$, $\mathbf{x}_{\text{ekey}}^{(b)} := (x_i^{(b)})_{i \in \mathcal{H}_{\text{ekey}}}$.

It can be verified that if an attack satisfies the previous notion of admissibility in the original work [15], such an attack will satisfy our notion of admissibility as well. Moreover, we can adapt naturally our admissibility from DMCFE to MCFE³ and also demonstrate that the prior DDH-based constructions for MCFE with deterministic encryption, for example from [2, 15, 17] to name a few, as well as an LWE-based construction for MCFE from [24] with slightly randomized encryption by Gaussian errors, cannot be proven secure in our new model by giving a concrete *admissible* attack, as already explained in Sect. 1.

3.3 Optimality of the New Admissibility: A Conceptual Challenge

After formulating a new admissibility condition, one natural question arises: *Is this the most suitable condition?* From a conceptual point of view, we want to prove that

For *certain* function classes, our admissibility cannot be relaxed, *i.e.* one cannot admit *some* non-admissible attack following our definition and still hope to be able to construct *some* specific efficient scheme that is provably secure.

Unsurprisingly, this poses a great definitional problem.

First of all, in all previous studies on (D)MCFE starting from [15], the admissibility concerns *adversaries* in the security game. Hence, if we want to prove the above claim, we need to consider *all* possible adversaries that can run non-admissible attacks and argue that they must be excluded. This is hard to argue rigorously, for example, what happens if a “dummy” adversary behaves in a non-admissible way but in the end outputs only a random guess for the challenge bit? Therefore, our very first step is to define the admissibility condition differently and take into account general *attacks* instead of adversaries. Afterwards, our optimality notion for an admissibility condition on attacks is stated that:

An admissibility is optimal for \mathcal{F} if we can construct a *passive* ppt distinguisher \mathcal{S} that receives *some* non-admissible attack coming from the queries of an adversary \mathcal{A} to a challenger Chall in the game for a DMCFE \mathcal{E} , uses only properties of \mathcal{F} , and devises a *generic* strategy to output the correct challenge bit with significant probability in the security game against *any* arbitrary DMCFE \mathcal{E}' .

Intuitively, \mathcal{S} passively observes the non-admissible queries in the attack from some specific interaction between \mathcal{A} against some specific scheme \mathcal{E} . Yet, these queries helps \mathcal{S} come up with a general approach to win significantly against any DMCFE scheme in a game that allows such non-admissible behaviors. This means it is impossible to prove security whenever this kind of behaviors is allowed. We formalize all these details in Definition 11 and Theorem 12, **Remark 14** elaborates more on the proof of our optimality claim. In the full version [25], our

³ The admissibility for MCFE is the particular condition when $\mathcal{H}_{\text{sk}} = [m]$ and thus $\mathbf{y}_{\text{sk}} = \mathbf{y}$, meaning the only deducible function is F itself.

admissibility’s optimality is verified for concrete most-studied function classes. Informally, we will explain in Sect. 4 the framework we proposed for arguing the optimality of an admissibility. Finally, the detailed admissibility condition for the class of inner products is given in Remark 15.

3.4 DMCFE for Inner Products with Refined Security Model

After introducing a new notion of admissibility in the security model for DMCFE and argue its optimality, we provide concrete constructions of DMCFE for inner products that are secure in this model. Our results are twofold. In Sect. 5.1 we give an intermediate construction where the new admissibility is translated in the case of inner-products together with the *complete* queries restriction (similar to condition 2 in previous works). In Sect. 6, we leverage this backbone construction from Sect. 5.1 to remove this complete queries restriction via a generic transformation as well as a concrete scheme with improved security.

In the following we highlight the main ideas of our backbone construction in Sect. 5.1. Our function class of interest is for computing inner products $\mathcal{F}^{\text{IP}} = \{F_{\mathbf{y}}\}$ and $F_{\mathbf{y}} : (\mathbb{Z}_q^*)^n \rightarrow \mathbb{Z}_q$ is defined as $F_{\mathbf{y}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{y} \rangle$. The parameter vector of $F_{\mathbf{y}}$ is simply $\mathbf{y} = (y_1, \dots, y_n) \in \mathbb{Z}_q^n$ and thus the number of parameters is the same as the dimension of the \mathbb{Z}_q -vector space for a prime q . Our construction relies on the notion of *Dual Pairing Vector Spaces* (DPVSes, see Sect. 2.2). We use DPVSes in the (additively written) bilinear group setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, \mathbf{e}, g_1, g_2, g_t)$. We sample n pairs of random dual bases $(\mathbf{H}_i, \mathbf{H}_i^*)_{i=1}^n$. Each client i will use their encryption key ek_i to encrypt the component x_i under some tag to get a ciphertext component $\text{ct}_{\text{tag},i}$, which is a vector of elements in \mathbb{G}_1 computed using \mathbf{H}_i . Accordingly, each secret key sk_i will be used by the DKShare in the decentralized key generation so as to generate a key component $\text{dk}_{\text{tag-f},i}$ for y_i under some tag-f, which is a vector of elements in \mathbb{G}_2 computed using \mathbf{H}_i^* . During decryption, the product $\text{ct}_{\text{tag},i} \times \text{dk}_{\text{tag-f},i}$ of vectors lying in dual bases will yield an element in \mathbb{G}_t of the form in the IPFE scheme by Agrawal, Libert, and Stehlé (ALS) [9]. We denote by $S = (s_1, \dots, s_n), U = (u_1, \dots, u_n)$ two vectors of secret scalars, intuitively which will be used in ALS ciphertext components $\llbracket s_i \omega + u_i \omega' + x_i \rrbracket$, where $\llbracket (\omega, \omega') \rrbracket \leftarrow \text{H}(\text{tag})$ comes from a full-domain hash function. In a centralized setting, such as the single-client scheme in [9] or the MCFE scheme in [15], the ALS key extraction provides $\langle S, \mathbf{y} \rangle$ and $\langle U, \mathbf{y} \rangle$ to be used in decryption.

Decentralizing ALS Key Extraction Under Separated Corruption. The first technical challenge is how to implement the ALS key extraction in a decentralized manner, because each key generator possessing y_i will not be able to compute $\langle S, \mathbf{y} \rangle$ and $\langle U, \mathbf{y} \rangle$ due to the lack of $(y_j)_{j \neq i}$. Our idea is to use $(s_i y_i, u_i y_i)$ in the i -th key components, masked by some randomness, then exploit the properties of products in DPVS that multiply facing coordinates together in order to “glue” this randomness to appropriate values in the i -th ciphertext component that enables correct decryption. More specifically, the components have the following form:

$$\begin{array}{l} \text{ct}_{\text{tag},i} \left(\cdots \left| \begin{array}{c} \omega p_i \\ s_i y_i \alpha_i + u_i y_i \gamma'_i \end{array} \right| \begin{array}{c} \omega' q_i \\ s_i y_i \gamma_i + u_i y_i \alpha_i \end{array} \left| \text{ALS-ciph} \right| \cdots \right)_{\mathbf{H}_i} ; \\ \text{dk}_{\text{tag-f},i} \left(\cdots \left| \begin{array}{c} \omega p_i \\ s_i y_i \alpha_i + u_i y_i \gamma'_i \end{array} \right| \begin{array}{c} \omega' q_i \\ s_i y_i \gamma_i + u_i y_i \alpha_i \end{array} \left| y_i \right| \cdots \right)_{\mathbf{H}_i^*} ; \end{array}$$

where ALS-ciph is the scalar in ALS ciphertext and (p_i, q_i) in $\text{ct}_{\text{tag},i}$ together with $(\alpha_i, \gamma_i, \gamma'_i)$ in $\text{dk}_{\text{tag-f},i}$ satisfy $p_i \alpha_i = \zeta_1, q_i \gamma_i = \zeta_2, q_i \alpha_i = \zeta_3, p_i \gamma'_i = \zeta_4$ and $\zeta_1, \zeta_2, \zeta_3, \zeta_4 \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$ are determined at setup time. However, the aforementioned local gluing technique is not enough, as it alone still reveals information about individual $x_i y_i$ from $\text{ct}_{\text{tag},i} \times \text{dk}_{\text{tag-f},i}$. A remedy is to put another layer of random secret sharings $\theta_i \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ into the partial key components such that $\sum_{i=1}^n \theta_i = 0$ so that only when n pairs of ciphertext-key components are combined together will we obtain the decrypted result. This new secret shares $(\theta_i)_i$ are randomized by $d_{\text{tag-f}} \leftarrow \text{H}(\text{tag-f})$ for each functional key, the newly randomized $(d_{\text{tag-f}} \theta_i)_i$ will behave indistinguishably from a random independent secret sharing of 0 under DDH. We refer to Sect. 5.1 and the transition $G_6 \rightarrow G_7$ in in the proof of Theorem 16 for more details.

HANDLING SEPARATED CORRUPTION. Each encryption key ek_i will contain information relevant to the basis \mathbf{H}_i so that client i can compute $\text{ct}_{\text{tag},i}$, meanwhile each key generator can use sk_i related to the *dual* basis \mathbf{H}_i^* for deriving the partial $\mathbf{k}_{i,\text{ipfe}}$. It appears that the contents of ek_i and sk_i belong to dual bases, independent for each i , and we handle their separated corruption by *basis changes* over $(\mathbf{H}_i, \mathbf{H}_i^*)$ in DPVS. We note that as soon as we program the basis change of one basis, we cannot control the change on its dual counterpart (defined by linear relations, see the full version [25]). To this end, our proofs can handle at best the scenario where one key type must be *statically* corrupted whereas the other’s corruption can be *dynamic*, otherwise the fact that for some i the keys ek_i and sk_i can be corrupted dynamically, in separate ways, can lead to inconsistency between basis changes. In particular, we use basis changes to program the master secret values $(s_i, u_i)_i$ as well as the secret sharings $(\theta_i)_i$, thus we want to program the changes on the dual bases \mathbf{H}_i^* . Consequently, we enforce static corruption on sk_i and perform those changes on i corresponding to honest sk_i ⁴.

Achieving New Admissibility by Embedding Revocation Mechanism into Components. The second technical challenge is to handle our new admissibility. In the prior weaker admissible condition introduced in [15], where $(\text{ek}_i, \text{sk}_i)$ are corrupted together, if $i \in [n]$ is corrupted then $\mathbf{x}_0^*[i] = \mathbf{x}_1^*[i]$. Putting forward the translation of our new admissibility in the functionality for inner products, the concrete conditions are: let $\Delta \mathbf{x}[i] := \mathbf{x}_0^*[i] - \mathbf{x}_1^*[i]$, then

1. For all $(\text{tag-f}, \mathbf{y}) \in \mathcal{Q}$, $\sum_{i \in (\mathcal{H}_{\text{skey}} \cup \mathcal{H}_{\text{ekey}})} \Delta \mathbf{x}[i] \mathbf{y}[i] = 0$.
2. For all $i \in \mathcal{C}_{\text{ekey}} \setminus \mathcal{C}_{\text{skey}}$, either $\mathbf{x}_0^*[i] = \mathbf{x}_1^*[i]$ or $\mathbf{y}[i] = 0$.
3. For all $i \in \mathcal{C}_{\text{skey}}$, $\mathbf{x}_0^*[i] = \mathbf{x}_1^*[i]$.

⁴ There are further involved technicalities to ensure that ek_i is constructed consistently, e.g. see the transition $G_7 \rightarrow G_8$ in the proof of Theorem 16.

As the main complication compared to [15, 17], when $i \in \mathcal{C}_{\text{ekey}} \setminus \mathcal{C}_{\text{skey}}$ it can be the case that $\Delta \mathbf{x}[i] \neq 0$ and $\mathbf{y}[i] = 0$. We want to ensure that even in this configuration the adversary cannot distinguish the i -th ciphertext components.

We interpret this situation in the language of *revocation*: if the adversary obtains the i -th key component $\text{dk}_{\text{tag-f},i}$ for $y_i := \mathbf{y}[i] = 0$, which is honestly generated as $i \in \mathcal{C}_{\text{ekey}} \setminus \mathcal{C}_{\text{skey}}$, then implicitly we are “revoking” the ability to learn information about the i -th challenge component using $\text{dk}_{\text{tag-f},i}$, even when the adversary can encrypt whatsoever using the corrupted ek_i , whose role now resembles a “public key” as in usual revocation systems. This leads us to the idea of embedding some sort of DDH-based revocation technique into each i -th component. We need to apply some revocation technique that is compatible with the bilinear group setting *and* the ALS ciphertext form, which is current employ at each component i in $\text{ct}_{\text{tag},i}$. We turn our attention to a recent work by Agrawal *et al.* [5], which builds public trace-and-revoke systems from standard assumptions and is particularly suitable for our objective because their constructions can be generically based on the DDH-based ALS IPFE. At a very high level, the decryption for m of the precedent scheme for revocation can be recasted as:

$$\frac{\text{ALS-IPFE.Dec}(\text{sk}_{id}, \text{ct})}{\langle \mathbf{x}_{id}, \mathbf{v}_R \rangle} = \frac{\langle \mathbf{x}_{id}, \mathbf{v}_R \cdot m \rangle}{\langle \mathbf{x}_{id}, \mathbf{v}_R \rangle} = m \quad (2)$$

where sk_{id} is the decryption given for an identity id , \mathbf{x}_{id} is some vector associated to id , and \mathbf{v}_R is derived from the revoked set R . With overwhelming probability, $\langle \mathbf{x}_{id}, \mathbf{v}_R \rangle \neq 0$ conditioned on $id \notin R$.

To adapt to our situation the division is translated to subtraction of coordinates and our “revoking” test depends only on a scalar y_i and whether $y_i = 0$ or not, the inner product become scalar multiplications in \mathbb{Z}_q^* . Consequently, we introduce extra coordinates in the DPVS bases $(\mathbf{H}_i, \mathbf{H}_i^*)$ to implement the aforementioned revocation technique, locally inside the vector’s components as follows:

$$\begin{array}{l} \text{ct}_{\text{tag},i} \left(\begin{array}{c} \omega p_i \\ s_i y_i \alpha_i + u_i y_i \gamma_i' \end{array} \middle| \begin{array}{c} \omega' q_i \\ s_i y_i \gamma_i + u_i y_i \alpha_i \end{array} \middle| \begin{array}{c} \text{ALS-ciph} - r_i v_i \\ y_i \end{array} \middle| \begin{array}{c} r_i t_i \\ y_i v_i / t_i \end{array} \middle| \cdots \middle| \text{rand} \right)_{\mathbf{H}_i}; \\ \text{dk}_{\text{tag-f},i} \left(\begin{array}{c} \omega p_i \\ s_i y_i \alpha_i + u_i y_i \gamma_i' \end{array} \middle| \begin{array}{c} \omega' q_i \\ s_i y_i \gamma_i + u_i y_i \alpha_i \end{array} \middle| \begin{array}{c} \text{ALS-ciph} - r_i v_i \\ y_i \end{array} \middle| \begin{array}{c} r_i t_i \\ y_i v_i / t_i \end{array} \middle| \cdots \middle| d_{\text{tag-f}} \theta_i \right)_{\mathbf{H}_i^*}; \end{array}$$

Using extra coordinates to contain $\llbracket (\text{ALS-ciph} - r_i v_i, r_i t_i) \rrbracket_1$ in $\text{ct}_{\text{tag},i}$ as well as $\llbracket (y_i, y_i v_i / t_i) \rrbracket_2$ in $\text{dk}_{\text{tag-f},i}$ helps us perform a “local” ALS+revocation decryption for component i , following the idea (2), when performing $\text{ct}_{\text{tag},i} \times \text{dk}_{\text{tag-f},i}$. Intuitively, our uniformly random scalar $r_i \xleftarrow{\$} \mathbb{Z}_q$ plays the role similar to that of \mathbf{x}_{id} in the blueprint from [5], that helps proving semantic security in the case of “revoked” $y_i = 0$ under random basis changes in DPVS using DDH. This probabilistic layer with $r_i \xleftarrow{\$} \mathbb{Z}_q$ allows to deal with corrupted encryption keys, even when $\mathbf{x}_0^*[i] \neq \mathbf{x}_1^*[i]$. This somehow covers public-key encryption. Our scheme is secure in the stronger security model under new admissibility and the complete queries restriction, *adaptively* in the challenges, with *dynamic* corruption of ek_i and *static* corruption of sk_i .

4 A Stronger Security Model for Decentralized Multi-Client Functional Encryption

This section presents a new security model for (D)MCFE, in which we refine the *admissibility* of adversaries in the security game and allow a more fine-grained corruption of keys. Following the line of works by Chotard *et al.* [15,17], such a notion of admissible adversaries is for excluding the attacks that are inevitable under which we cannot prove security. Our objective is to define the admissible condition in a way that excludes as few attacks as possible, and as soon as such condition is weakened, there is an unconditional generic attack to trivially win the security game against *any* concrete scheme. First of all, Definition 3 formalizes the terminologies of parameters of a function and deducible functions/inputs.

Definition 3 (Deducible inputs and functions). Fix $\lambda \in \mathbb{N}$ and denote by \mathcal{F}_λ a family of n -ary functions indexed by λ , with domain $\mathcal{D}_{\lambda,1} \times \cdots \times \mathcal{D}_{\lambda,n}$ and range \mathcal{R}_λ , where $n = n(\lambda) \in \mathbb{N}$ is a function. Let $\text{Param}_1, \dots, \text{Param}_m$ denote m sets of parameters for functions in \mathcal{F}_λ , where $m = m(\lambda) \in \mathbb{N}$ is a function. Suppose there exists a deterministic encoding $\mathfrak{p} : \mathcal{F}_\lambda \rightarrow \text{Param}_1 \times \cdots \times \text{Param}_m$, that maps a function $F_{\mathbf{y}} \in \mathcal{F}_\lambda$ to its parameters $\mathbf{y} \in \text{Param}_1 \times \cdots \times \text{Param}_m$. Let $\mathcal{H}_{\text{inp}} \subseteq [n], \mathcal{H}_{\text{func}} \subseteq [m]$ and suppose we are given $\mathbf{x}_{\text{inp}} \in (\mathcal{D}_{\lambda,i})_{i \in \mathcal{H}_{\text{inp}}}$ and $\mathbf{y}_{\text{func}} \in (\text{Param}_j)_{j \in \mathcal{H}_{\text{func}}}$ as lists of inputs and parameters that are indexed by $\mathcal{H}_{\text{inp}}, \mathcal{H}_{\text{func}}$ respectively.

A vector $\mathbf{z} \in \mathcal{D}_{\lambda,1} \times \cdots \times \mathcal{D}_{\lambda,n}$ is said to be deducible from \mathbf{x}_{inp} if $\forall i \in \mathcal{H}_{\text{inp}} : \mathbf{z}[i] = \mathbf{x}_{\text{inp}}[i]$. A function G is said to be deducible from \mathbf{y}_{func} if for all $i \in \mathcal{H}_{\text{func}}$, we have $\mathfrak{p}(G)[i] = \mathbf{y}_{\text{func}}[i]$.

Remark 4. If $\mathbf{y}_{\text{func}} = \mathbf{y}$, for the parameter \mathbf{y} of some function $F_{\mathbf{y}}$, then the only function deducible from \mathbf{y}_{func} is $F_{\mathbf{y}}$ itself. In the DMCFE setting, there will be situations with non-trivial \mathbf{y}_{func} where $\mathcal{H}_{\text{func}} \subsetneq [m]$. For concrete function classes in our construction, we focus on the class to compute inner products $\mathcal{F}^{\text{IP}} = \{F_{\mathbf{y}}\}$ and $F_{\mathbf{y}} : (\mathbb{Z}_q^*)^n \rightarrow \mathbb{Z}_q$ that is defined as $F_{\mathbf{y}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{y} \rangle$. For inner products, the parameters of a function $F_{\mathbf{y}} \in \mathcal{F}^{\text{IP}}$ can be precisely defined to be $\mathfrak{p}(F_{\mathbf{y}}) := \mathbf{y} \in \mathbb{Z}_q^n$ and the number of parameters m is equal to the number of arguments n . When \mathcal{F}_λ is clear from context, we omit the subscript λ .

We now recall the notion of *decentralized multi-client functional encryption* (DMCFE) whose syntax is given below.

Definition 5 (Decentralized Multi-Client Functional Encryption). A decentralized multi-client functional encryption (DMCFE) scheme for a function class \mathcal{F} consists of five algorithms

$$(\text{Setup}, \text{DKShare}, \text{DKeyComb}, \text{Enc}, \text{Dec})$$

that are defined below:

- Setup**(1^λ): Given as input a security parameter λ and $n = n(\lambda), m = m(\lambda)$, generate in a possibly interactive manner n encryption keys $(\mathbf{ek}_i)_{i \in [n]}$ as well as m secret keys $(\mathbf{sk}_j)_{j \in [m]}$ where $m, n : \mathbb{N} \rightarrow \mathbb{N}$ are functions.
- Enc**($\mathbf{ek}_i, \text{tag}, x_i$): Given as inputs an encryption key \mathbf{ek}_i , a message $x_i \in \mathcal{D}_{\lambda, i}$, and a tag tag , output a ciphertext $\text{ct}_{\text{tag}, i}$.
- DKShare**($\mathbf{sk}_j, \text{tag-f}, y_j$): Given a secret key \mathbf{sk}_j , a tag $\text{tag-f} \in \text{Tag}$, and the j -th parameter y_j , output a partial functional key $\mathbf{dk}_{\text{tag-f}, j}$.
- DKeyComb**($(\mathbf{dk}_{\text{tag-f}, j})_{j \in [m]}, \text{tag-f}, F$): Given a tag tag-f together with a function F and m partial functional keys $\mathbf{dk}_{\text{tag-f}, j}$ for the parameters $\mathbf{p}(F)$, output the functional key $\mathbf{dk}_{\text{tag-f}, F}$.
- Dec**($\mathbf{dk}_{\text{tag-f}, F}, \mathbf{c}$): Given the functional decryption key $\mathbf{dk}_{\text{tag-f}, F}$ and a list of ciphertexts $\mathbf{c} := (\text{ct}_{\text{tag}, i})_{i=1}^n$ of length n , output an element in \mathcal{R}_λ or an invalid symbol \perp .

We make the assumption that all public parameters are included in the secret keys and the encryption keys, as well as the (partial) functional decryption key.

Correctness. We require that for sufficiently large $\lambda \in \mathbb{N}$, for all $\text{tag}, \text{tag-f} \in \text{Tag}$, for all $F \in \mathcal{F}$, $(x_i)_{i \in [n]} \in \mathcal{D}_{\lambda, 1} \times \cdots \times \mathcal{D}_{\lambda, n}$ and

$$\begin{aligned} \mathbf{sk}_j, (\mathbf{ek}_i)_{i \in [n]} &\leftarrow \text{Setup}(1^\lambda); \mathbf{dk}_{\text{tag-f}, j} \leftarrow \text{DKShare}(\mathbf{sk}_j, \text{tag-f}, y_j)_{j \in [m]} ; \\ \mathbf{dk}_{\text{tag-f}, F} &\leftarrow \text{DKeyComb}((\mathbf{dk}_{\text{tag-f}, j})_j, \text{tag-f}, F); (\text{ct}_{\text{tag}, i})_i \leftarrow (\text{Enc}(\mathbf{ek}_i, \text{tag}, x_i))_i \end{aligned}$$

where $i \in [n]$ and $j \in [m]$, the following holds with overwhelming probability:

$$\text{Dec}(\mathbf{dk}_{\text{tag-f}, F}, (\text{ct}_{\text{tag}, i})_{i=1}^n) = F(x_1, \dots, x_n) \text{ when } F(x_1, \dots, x_n) \neq \perp^7 \quad (3)$$

where $F : \mathcal{D}_{\lambda, 1} \times \cdots \times \mathcal{D}_{\lambda, n} \rightarrow \mathcal{R}_\lambda$ and the probability is taken over the random coins of algorithms.

Security. We follow the approach in the work by Chotard *et al.* [15] so as to define the security game with oracles **Initialize**, **Corrupt**, **LoR**, **Enc**, **DKeyGenShare**, and **Finalize**. We recall that the oracle **Enc** is necessary for a simpler notion of one challenge, while retaining an equivalence to the multi-challenge notion using a hybrid argument shown in Lemma 8.

The adversary is also able to corrupt *separately* the secret key \mathbf{sk}_j of any key-generator j as well as the encryption key \mathbf{ek}_i of any client i , which is done via requests (i, skey) or (j, ekey) to the oracle **Corrupt**, respectively. We need to exclude trivial attacks that can be mounted in the security experiment. Those restrictions are encompassed in the notion of *admissibility*, which is extended from similar notions in the works of [15, 17].

In a nutshell, Definition 6 gives the definition of admissibility, generalizing the admissibility condition that has been consistently used since the seminal work of Chotard *et al.* [15]. We refer to Sect. 3 for a high-level discussion. In the subsequent Sect. 4.1, we give the full formal treatment to prove the *optimality* of our admissibility condition in Definition 6. The successive security analyses of our DMCFE schemes rely crucially on this definition, translated for the concrete class of inner product in Remark 15.

Definition 6 (Admissibility condition). Let \mathcal{A} be a ppt adversary and let

$$\mathcal{E} = (\text{Setup}, \text{DKShare}, \text{DKeyComb}, \text{Enc}, \text{Dec})$$

be a DMCFE scheme for a function class \mathcal{F} set up w.r.t $\lambda \in \mathbb{N}$. In **Finalize**, considering the queries $(\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \mathcal{C}_{\text{skey}}, \mathcal{C}_{\text{ekey}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}, \{(\mathbf{x}, \text{tag}^{(k)})\})$, we say that the attack corresponding to these queries is **NOT** admissible if the following is satisfied

There exist $\text{tag}, \text{tag-f} \in \text{Tag}$, a function $F \in \mathcal{F}$ having parameters $\mathbf{y} \in \text{Param}_1 \times \dots \times \text{Param}_m$, two challenges $(x_i^{(0)}, x_i^{(1)})_{i \in [n]}$ such that $(\text{tag-f}, F) \in \mathcal{Q}$ is queried to **DKeyGenShare**, $((x_i^{(0)}, x_i^{(1)})_{i \in [n]}, \text{tag})$ is queried to **LoR** and there exists a pair $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)})$ deducible from $(\mathbf{x}_{\text{ekey}}^{(0)}, \mathbf{x}_{\text{ekey}}^{(1)})$, a function G deducible from \mathbf{y}_{skey} satisfying

$$G(\mathbf{z}^{(0)}) \neq G(\mathbf{z}^{(1)}) \text{ ,} \tag{4}$$

where we define $\mathbf{y}_{\text{skey}} := (\mathbf{y}[i])_{i \in \mathcal{H}_{\text{skey}}}$ and for $b \in \{0, 1\}$, $\mathbf{x}_{\text{ekey}}^{(b)} := (x_i^{(b)})_{i \in \mathcal{H}_{\text{ekey}}}$.

Otherwise, we say that the aforementioned attack is admissible.

Definition 7 (IND-security for DMCFE). A DMCFE scheme

$$\mathcal{E} = (\text{Setup}, \text{DKShare}, \text{DKeyComb}, \text{Enc}, \text{Dec})$$

for the function class $\mathcal{F} = \{F_\lambda\}_{\lambda \in \mathbb{N}}$ is **xx**-secure if for all ppt adversaries \mathcal{A} , and for all sufficiently large $\lambda \in \mathbb{N}$, the following probability is negligible

$$\text{Adv}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{xx}}(1^\lambda) := \left| \Pr[\text{Expr}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{xx}}(1^\lambda) = 1] - \frac{1}{2} \right| \text{ .}$$

The game $\text{Expr}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{xx}}(1^\lambda)$ is depicted in Fig. 1. The security level indicator **xx** can be: **dmc-ind-cpa** to indicate IND-security with adaptive challenges, dynamic corruption of ekey, and dynamic corruption of skey; **dmc-sel-ind-cpa** to indicate selective IND-security with selective challenges, dynamic corruption of ekey, and dynamic corruption of skey; **dmc-stat-ind-cpa** to indicate static IND-security with adaptive challenges, static corruption of ekey, and static corruption of skey⁵; **dmc-ind-cpa-1chal** indicate one-time IND-security with one adaptive challenge tag, dynamic corruption of ekey, and dynamic corruption of skey. The probability is taken over the random coins of \mathcal{A} and the algorithms.

⁵ In addition, we can allow dynamic corruption on one type but static corruption on the other type of keys, such as **dmc-stat-sk-ind-cpa** to indicate *partially static IND-security* with adaptive challenges, dynamic corruption of ekey, and static corruption of skey.

Lemma 8 allows us to concentrate on the notion of one-time IND-security for our DMCFE constructions. The proof is a standard hybrid argument and we give it in the full version [25] for completeness.

Lemma 8. *Let $\mathcal{E} = (\text{Setup}, \text{DKShare}, \text{DKeyComb}, \text{Enc}, \text{Dec})$ be a DMCFE scheme for the function class \mathcal{F} . If \mathcal{E} is one-time IND-secure, then \mathcal{E} is IND-secure.*

4.1 Optimality of Admissibility as per Definition 6

In this section, we demonstrate that our notion of admissibility in Definition 6 is capturing all trivial attacks against DMCFE schemes for non-trivial function classes, which include the class of inner-product and quadratic functionalities. The high-level plan is given below.

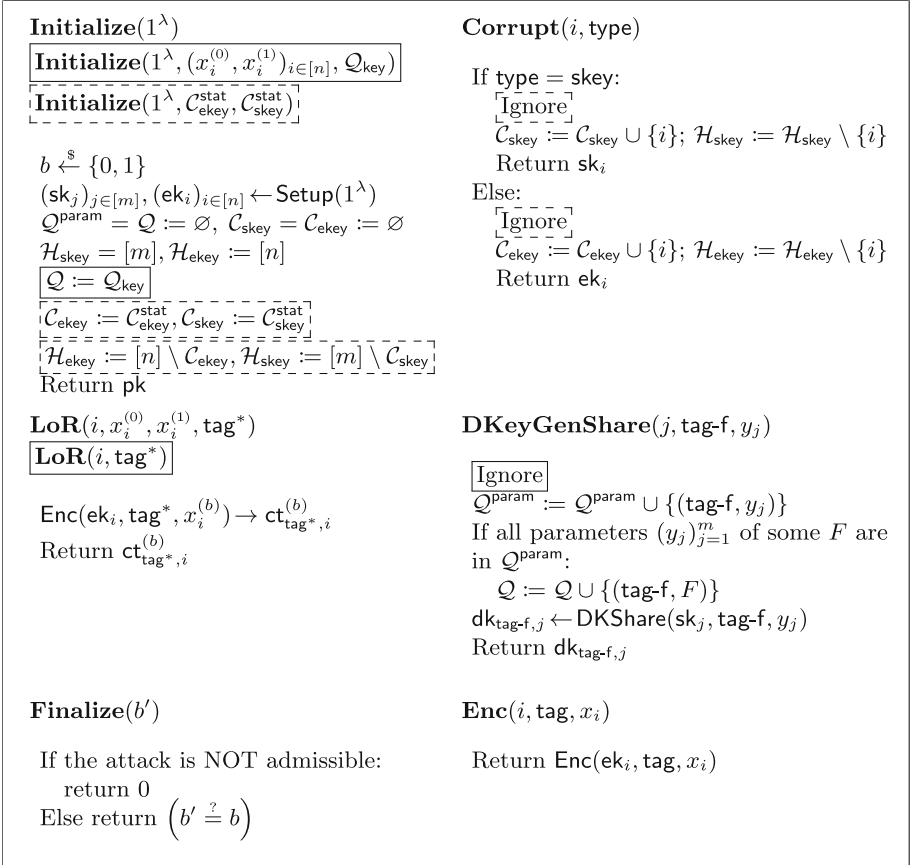


Fig. 1. The security games $\text{Exp}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{dmc-ind-cpa}}(1^\lambda)$, $\text{Exp}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{dmc-sel-ind-cpa}}(1^\lambda)$, and $\text{Exp}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{dmc-stat-ind-cpa}}(1^\lambda)$ for Definition 7 and 17. The admissibility condition is defined in Definition 6.

PROVING OPTIMALITY. The general idea on defining the *optimality* of an admissibility condition can be revisited in Sect. 3.3. We now explain briefly how one can show that an admissibility condition is optimal following what we try to define. First and foremost, a notion of optimality makes sense when we consider only certain functionalities and not any arbitrary class of functions. For example, for a general functionality, the adversary's admissibility as defined in Definition 6 might not be *efficiently* decidable. Roughly speaking, **Finalize** may have to go through *all* $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)})$ deducible from $(\mathbf{x}_{\text{ekey}}^{(0)}, \mathbf{x}_{\text{ekey}}^{(1)})$ and *all* G deducible from $\mathbf{y}_{\text{skkey}}$ so as to check relation (4). Therefore, we want to focus on classes for which the admissibility can be decided efficiently by **Finalize**, at least for the sake of having an efficient challenger in the security game.

In addition, we require a further property of the functionality under consideration: in view of the admissibility check (4), the deduction of $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)})$ from $(\mathbf{x}_{\text{ekey}}^{(0)}, \mathbf{x}_{\text{ekey}}^{(1)})$ and of a function G from $\mathbf{y}_{\text{skkey}}$ can be done efficiently. We coin this property *fixed-component distinguishability*. In summary, we restrain the optimality evaluation to classes that are (1) fixed-component distinguishable and (2) for which the admissibility is efficiently decidable. In the full version [25], we prove that both most studied functionalities for inner products and quadratic evaluations satisfy properties (1) and (2).

The core of our reasoning that an admissibility condition is optimal comprises building a ppt distinguisher, which can exert a *non-admissible* attack, to trivially win significantly the security game against *any* DMCFE scheme. We recall that Definition 6 views attacks as ensembles of queries made by some adversary in its security game. Because the class allows deciding efficiently the admissibility, our distinguisher can efficiently determine which query in the attack will violate the check (4), and thanks to the fixed-component distinguishability, the triplet $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)}, G)$ can be concretely reconstructed in an efficient manner.

In the end, facing any DMCFE challenger that allows the foregoing non-admissible behaviour, our distinguisher can simply use $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)}, G)$ to trivially win the game. This means that whenever we allow a non-admissible attack, or in other words whenever we try to relax Definition 6, no DMCFE scheme can be proved secure due to the existence of the above distinguisher.

To begin our formal treatment, we restrain our attention to particular function classes that satisfy certain properties.

Definition 9 (Fixed-component distinguishable classes). Fix $\lambda \in \mathbb{N}$ and denote by $\mathcal{F}_\lambda = \{F : \mathcal{D}_{\lambda,1} \times \cdots \times \mathcal{D}_{\lambda,n} \rightarrow \mathcal{R}_\lambda\}$ a family of n -ary functions indexed by λ having m parameters, where $m = m(\lambda)$, $n = n(\lambda)$ are functions.

For $F \in \mathcal{F}_\lambda$, a triple $(\mathbf{x}_{\text{inp}}^{(0)}, \mathbf{x}_{\text{inp}}^{(1)}, \mathcal{H}_{\text{inp}})$, where $\mathbf{x}_{\text{inp}}^{(b)} \in \prod_{i \in \mathcal{H}_{\text{inp}}} \mathcal{D}_{\lambda,i}$ for $b \in \{0, 1\}$, is said to be distinguishing F_λ with fixed components if there exists a deducible pair $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)}) \in \prod_{i \in [n]} \mathcal{D}_{\lambda,i}$ such that $F_\lambda(\mathbf{z}^{(0)}) \neq F_\lambda(\mathbf{z}^{(1)})$ where

$$\begin{cases} \mathbf{z}^{(b)}[i] = \mathbf{x}^{(b)}[i] \text{ for } b \in \{0, 1\}, i \in \mathcal{H}_{\text{inp}} \\ \mathbf{z}^{(0)}[i] = \mathbf{z}^{(1)}[i] \ \forall i \in [n] \setminus \mathcal{H}_{\text{inp}} \end{cases}.$$

A function F is said to be fixed-component distinguishable if there exists a triple distinguishing F with fixed components and a ppt Turing machine that, given as input this triple, outputs the corresponding deducible pair.

A function class \mathcal{F}_λ is fixed-component distinguishable if for all $F \in \mathcal{F}_\lambda$ with parameters $\mathbf{p} := \mathbf{p}(F)$, there exists a fixed-component distinguishable function G deducible from $(\mathbf{p}[i])_{i \in \mathcal{H}_{\text{func}}}$ for some $\mathcal{H}_{\text{func}} \subseteq [m]$, and a ppt Turing machine that, given as inputs $(F, \mathcal{H}_{\text{func}})$, outputs G .

Remark 10. We remark that a function class \mathcal{F} is fixed-component distinguishable does *not* necessarily imply that the admissibility from Definition 6 for \mathcal{F} can be efficiently decided. Roughly speaking, given a function among the adversary's queries, the ppt Turing machine from fixed-component distinguishability will output *some* deducible function G for which one can test the admissible condition (4) efficiently. But that is not enough, as to decide the admissibility of an attack, we need to check *all* such deducible functions and there is no further guarantee in the case of general functionalities that we can do this check efficiently. In the concrete cases of inner products and quadratic functions, the check over all such deducible functions can be done efficiently by using their linear properties, see the full version [25] for more details.

We now define what means for an admissibility to be *optimal* for a function class \mathcal{F} . For simplicity, we can consider the one-challenge setting thanks to Lemma 8.

Definition 11. Let $\lambda \in \mathbb{N}$ and denote by \mathcal{F} a family of $n(\lambda)$ -ary functions indexed by λ , with $m(\lambda)$ parameters for each member of \mathcal{F} . An admissibility condition $\text{adm}(\cdot)$ is optimal for \mathcal{F} if there exists a ppt distinguisher \mathcal{S} so that for all non-admissible

$$(\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \mathcal{C}_{\text{skey}}, \mathcal{C}_{\text{ekey}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}, \{(\mathbf{x}, \text{tag}^{(k)})\})$$

of some ppt adversary \mathcal{A} and against a DMCFE \mathcal{E} for \mathcal{F} in a security experiment $\text{Expr}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}$ given in Fig. 1, we have

$$\begin{aligned} \Pr[\mathcal{S}((\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \mathcal{C}_{\text{skey}}, \mathcal{C}_{\text{ekey}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}, \{(\mathbf{x}, \text{tag}^{(k)})\})) = b] & : b \leftarrow \text{Chall}(\text{rand}_{\text{Chall}}) \\ & \geq \frac{1}{\text{poly}(\lambda)} \end{aligned}$$

where $(\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \mathcal{C}_{\text{skey}}, \mathcal{C}_{\text{ekey}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}, \{(\mathbf{x}, \text{tag}^{(k)})\})$ is well-defined at the time of **Finalize** and $b \leftarrow \text{Chall}(\text{rand}_{\text{Chall}})$ means the challenger Chall uses the bit b in $\text{Expr}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}$.

We now state our main theorem for the optimality of our admissibility.

Theorem 12. Let \mathcal{F} be a function class that has efficient decidability for admissibility and is fixed-component distinguishable. Then, our admissibility condition as defined in Definition 6 is optimal for \mathcal{F} .

Proof (Of Theorem 12). Without loss of generality, we consider the one-challenge notion. We need to prove that: there exists a ppt distinguisher \mathcal{S} so that for any non-admissible $(\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \mathcal{C}_{\text{skey}}, \mathcal{C}_{\text{ekey}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}, \{(\mathbf{x}, \text{tag}^{(k)})\})$ of some DMCFE \mathcal{E} for \mathcal{F} and some ppt adversary \mathcal{A} in a security experiment $\text{Exp}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{xx}}$ given in Fig. 1, we have

$$\begin{aligned} \Pr [\mathcal{S}((\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \mathcal{C}_{\text{skey}}, \mathcal{C}_{\text{ekey}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}, \{(\mathbf{x}, \text{tag}^{(k)})\})) = b] & : b \leftarrow \text{Chall}(\text{rand}_{\text{Chall}}) \\ & \geq \frac{1}{\text{poly}(\lambda)}. \end{aligned}$$

Let $\mathcal{E}^{\text{abs}} = (\text{Setup}^{\text{abs}}, \text{DKShare}^{\text{abs}}, \text{DKeyComb}^{\text{abs}}, \text{Enc}^{\text{abs}}, \text{Dec}^{\text{abs}})$ be an abstract DMCFE for \mathcal{F} that satisfies the correctness relation (3). We describe the distinguisher \mathcal{S} as follows:

1. The distinguisher \mathcal{S} parses

$$(\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \mathcal{C}_{\text{skey}}, \mathcal{C}_{\text{ekey}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}, \{(\mathbf{x}, \text{tag}^{(k)})\})$$

then use \mathcal{E}^{abs} and $(\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\})$ for abstracting the key components to obtain $\{(\text{dk}_{\text{tag-f}, F, j}^{\text{abs}})_{j \in [m]}\}_{(\text{tag-f}, F) \in \mathcal{Q}}$, the challenge ciphertext components to obtain $(\text{ct}_{\text{tag}, i}^{\text{abs}})_{i \in [n]}$ for each $\{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}$, and the encryption responses to obtain $(\text{ct}_i^{\text{abs}, (k)})_{i \in [n]}$. If there are corrupted keys sk_j or ek_i queried by \mathcal{A} , they will also be replaced by their abstracted counterparts sk_j^{abs} or ek_i^{abs} . In the following \mathcal{S} only needs the abstract DMCFE \mathcal{E}^{abs} for \mathcal{F} , no matter what the details of the concrete \mathcal{E} are.

2. If there exists $(\text{tag-f}, F) \in \mathcal{Q}$ such that $F(\mathbf{x}_0^*) \neq F(\mathbf{x}_1^*)$, \mathcal{S} combines the key components of $(\text{tag-f}, F)$, decrypts the challenge ciphertext components, and outputs 1 if and only if the result is $F(\mathbf{x}_1^*)$. All algorithms come from $\mathcal{E}^{\text{abs}} = (\text{Setup}^{\text{abs}}, \text{DKShare}^{\text{abs}}, \text{DKeyComb}^{\text{abs}}, \text{Enc}^{\text{abs}}, \text{Dec}^{\text{abs}})$. Else, in the following we assume that $F(\mathbf{x}_0^*) = F(\mathbf{x}_1^*)$ for all $(\text{tag-f}, F) \in \mathcal{Q}$.
3. Because this is a non-admissible attack, \mathcal{S} uses the efficient decidability of \mathcal{F} to find $(\text{tag-f}, F) \in \mathcal{Q}$, whose parameters is $\mathbf{y} := \mathbf{p}(F)$, so that: there exists a pair $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)})$ deducible from $(\mathbf{x}_{\text{ekey}}^{(0)}, \mathbf{x}_{\text{ekey}}^{(1)})$, a function G deducible from \mathbf{y}_{skey} satisfying

$$G(\mathbf{z}^{(0)}) \neq G(\mathbf{z}^{(1)}),$$

where $\mathbf{y}_{\text{skey}} := (\mathbf{y}[i])_{i \in \mathcal{H}_{\text{skey}}}$ and $\mathbf{x}_{\text{ekey}}^{(0)} := (x_i^{(0)})_{i \in \mathcal{H}_{\text{ekey}}}$, $\mathbf{x}_{\text{ekey}}^{(1)} := (x_i^{(1)})_{i \in \mathcal{H}_{\text{ekey}}}$. We remark that finding F can be done efficiently in \mathcal{Q} because the current attack comes from the execution of some ppt adversary \mathcal{A} , which implies the size of \mathcal{Q} is polynomially bounded.

4. Because \mathcal{F} is fixed-component distinguishable (see Definition 9), using F and $\mathcal{C}_{\text{skey}}$, \mathcal{S} can efficiently find a function G deducible from \mathbf{y}_{skey} such that G is fixed-component distinguishable.
5. Thanks to the fixed-component distinguishability of G , using $(\mathbf{x}_{\text{ekey}}^{(0)}, \mathbf{x}_{\text{ekey}}^{(1)})$ and $\mathcal{H}_{\text{ekey}}$, the pair $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)})$ can be found efficiently by \mathcal{S} .
6. The distinguisher \mathcal{S} then uses the corrupted keys $(\text{ek}_i^{\text{abs}})_{i \in \mathcal{C}_{\text{ekey}}}$ to compute new ciphertext components $(\tilde{\text{ct}}_{\text{tag}, i}^{\text{abs}})_{i=1}^n$ of $\mathbf{z}^{(b)}$ implicitly, using $(\text{ct}_{\text{tag}, i}^{\text{abs}})_{i \in \mathcal{H}_{\text{ekey}}}$

- for the challenge $(\mathbf{x}_b^*[i])_{i \in \mathcal{H}_{\text{ekey}}}$, and using Enc^{abs} to encrypt $(\mathbf{z}^{(b)}[i])_{i \in \mathcal{C}_{\text{ekey}}}$ using $(\mathbf{ek}_i^{\text{abs}})_{i \in \mathcal{C}_{\text{ekey}}}$.
7. Next, \mathcal{S} uses the corrupted keys $(\mathbf{sk}_i^{\text{abs}})_{i \in \mathcal{C}_{\text{skey}}}$ to compute new key components $(\tilde{\mathbf{dk}}_{\text{tag-f},G,i}^{\text{abs}})_{i=1}^n$ of G implicitly, using $(\mathbf{dk}_{\text{tag-f},F,i}^{\text{abs}})_{i \in \mathcal{H}_{\text{skey}}}$, and using $\text{DKShare}^{\text{abs}}$ to derive $(\tilde{\mathbf{dk}}_{\text{tag-f},F,i}^{\text{abs}})_{i \in \mathcal{C}_{\text{skey}}}$ from $(\mathbf{p}(G)[i])_{i \in \mathcal{C}_{\text{skey}}}$.
 8. Finally, \mathcal{S} uses $\text{DKeyComb}^{\text{abs}}$ to combine the newly generated key components $(\tilde{\mathbf{dk}}_{\text{tag-f},i}^{\text{abs}})_{i=1}^n$. Then \mathcal{S} decrypts the newly generated challenge ciphertext $(\tilde{\mathbf{ct}}_{\text{tag},i}^{\text{abs}})_{i \in [n]}$ using the abstract algorithm Dec^{abs} , the adversary outputs 1 if and only if the result is equal to $G(\mathbf{z}^{(1)})$.

In the end, \mathcal{S} outputs 1 if and only if $(\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \mathcal{C}_{\text{skey}}, \mathcal{C}_{\text{ekey}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}, \{(\mathbf{x}, \text{tag}^{(k)})\})$ comes from an execution of any \mathcal{A} against Chall of \mathcal{E} in which Chall picks 1 as the challenge bit. This concludes the proof. \square

Remark 13. The abstract object \mathcal{E}^{abs} in the proof of Theorem 12 are used *only* in our formal proofs of the optimality for our admissible condition. In the concrete constructions of DMCFE, no such abstract objects are needed, as the admissibility are examined via concrete tests over the adversary’s queries in the security game. For instance, see the full version [25] for the cases of linear and quadratic functions.

Remark 14. The generic distinguisher \mathcal{S} in Theorem 12 is *weak* in the sense that all it has is a non-admissible attack with the corresponding

$$(\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \mathcal{C}_{\text{skey}}, \mathcal{C}_{\text{ekey}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}, \{(\mathbf{x}, \text{tag}^{(k)})\})$$

determined by \mathcal{A} ’s behaviour during the security game, *not depending on* the concrete implementation of \mathcal{E} . However, thanks to the non-admissibility of the given attack and the fixed-component distinguishability of the function class, \mathcal{S} can still output the correct challenge bit, in the security against *any* DMCFE scheme. This means that as soon as we allow some non-admissible behaviour, where the concrete descriptions of \mathcal{A} and \mathcal{E} can be arbitrary as long as this behaviour stays the same, there is no hope in proving security regardless of the specific implementation of \mathcal{E} . Equivalently, our Definition 6 that excludes exactly these non-admissible attacks cannot be enlarged in any sense and captures the most attacks against which we can prove security. Last but not least, we see clearly the role of the abstract DMCFE \mathcal{E}^{abs} : it abstracts out the concrete details of *some* specific scheme \mathcal{E} from which calculations over the non-admissible queries can be done, and return the “black-boxed” data that obey the correctness of DMCFE schemes for \mathcal{F} .

Remark 15. As a corollary the admissibility’s optimality for the class of inner products (including for polynomially bounded ranges - see the full version [25] for more details), we have specific conditions for admissible attacks in this case:

1. For all vectors $(\mathbf{x}_0^*, \mathbf{x}_1^*)$ that is queried to LoR , for all $(\text{tag-f}, \mathbf{y}) \in \mathcal{Q}$, $\sum_{i \in \mathcal{H}} \Delta \mathbf{x}[i] \mathbf{y}[i] = 0$ where $\Delta \mathbf{x}[i] = \mathbf{x}_0^*[i] - \mathbf{x}_1^*[i]$, where $\mathcal{H} := \mathcal{H}_{\text{ekey}} \cap \mathcal{H}_{\text{skey}}$.

2. For all vectors $(\mathbf{x}_0^*, \mathbf{x}_1^*)$ that is queried to **LoR**, for all $(\text{tag-f}, \mathbf{y}) \in \mathcal{Q}$, for all $i \in \mathcal{C}_{\text{key}} \setminus \mathcal{C}_{\text{skey}}$, either $\mathbf{x}_0^*[i] = \mathbf{x}_1^*[i] = 0$ or $\mathbf{y}[i] = 0$.
3. For all vectors $(\mathbf{x}_0^*, \mathbf{x}_1^*)$ that is queried to **LoR**, for all $(\text{tag-f}, \mathbf{y}) \in \mathcal{Q}$, for all $i \in \mathcal{C}_{\text{skey}}$, $\mathbf{x}_0^*[i] = \mathbf{x}_1^*[i]$.

5 DMCFE for Inner Products with Stronger Security Against Complete Queries

5.1 Construction

This section presents a *decentralized* multi-client FE scheme, as defined in Sect. 4, for the function class $\mathcal{F}_{B,B'}^{\text{IP,poly}}$ and $F_{\mathbf{y}} : (\mathbb{Z}_q^*)^n \rightarrow \mathbb{Z}_q$ is defined as $F_{\mathbf{y}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{y} \rangle$ where $\|\mathbf{x}\|_\infty < B$ and $\|\mathbf{y}\|_\infty < B'$, where $B, B' = \text{poly}(\lambda) \in \mathbb{N}$ are polynomials. The high-level ideas are discussed in Sect. 3.4. As discussed in Remark 4, the parameter vector of $\mathcal{F}_{B,B'}^{\text{IP,poly}}$ is simply \mathbf{y} of size n . Hence the number of secret keys and of encryption keys are equal to n . Our admissibility is also optimal for $\mathcal{F}_{B,B'}^{\text{IP,poly}}$, see the full version [25]. We need a full-domain hash function $\mathbf{H}_1 : \text{Tag} \rightarrow \mathbb{G}_1^2$, where Tag denotes the set of tags used for ciphertext components and functional key components. In addition, we also need a hash function $\mathbf{H}_2 : \text{Tag} \times \mathbb{Z}_q^n \rightarrow \mathbb{Z}_q$.

We are in the bilinear group setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, \langle \cdot, \cdot \rangle, \mathbf{e}, q)$ and $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ are written additively. The details of our DMCFE scheme go as follows:

Setup(1^λ): Choose n pairs of dual orthogonal bases $(\mathbf{H}_i, \mathbf{H}_i^*)$ for $i \in [n]$, where $(\mathbf{H}_i, \mathbf{H}_i^*)$ is a pair of dual bases for $(\mathbb{G}_1^6, \mathbb{G}_2^6)$. We denote the basis changing matrices for $(\mathbf{H}_i, \mathbf{H}_i^*)$ as (H_i, H_i') :

$$(\mathbf{H}_i = H_i \cdot \mathbf{T}; \mathbf{H}_i^* = H_i' \cdot \mathbf{T}^*)_{i \in [n]}$$

where $H_i, H_i' \in \mathbb{Z}_q^{6 \times 6}$ and $(\mathbf{T} = [[I_6]_1], \mathbf{T}^* = [[I_6]_2])$ are canonical bases of $(\mathbb{G}_1^6, \mathbb{G}_2^6)$, for the identity matrix I_6 . Sample two full-domain hash functions $\mathbf{H}_1 : \text{Tag} \rightarrow \mathbb{G}_1^2$ and $\mathbf{H}_2 : \text{Tag} \times \mathbb{Z}_q^n \rightarrow \mathbb{Z}_q$. We recall that interactions are involved only in this Setup phase. For each $i \in [n]$, we write

$$\mathbf{H}_i = (\mathbf{h}_{i,1}, \mathbf{h}_{i,2}, \dots, \mathbf{h}_{i,6}) \quad \mathbf{H}_i^* = (\mathbf{h}_{i,1}^*, \mathbf{h}_{i,2}^*, \dots, \mathbf{h}_{i,6}^*)$$

and sample $S, U, V, T \stackrel{\$}{\leftarrow} (\mathbb{Z}_q^*)^n$ where $S = (s_1, \dots, s_n)$, $U = (u_1, \dots, u_n)$, $V = (v_1, \dots, v_n)$, $T = (t_1, \dots, t_n)$. Sample $\theta_i \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$ such that $\sum_{i=1}^n \theta_i = 0$. Then, sample $\zeta_1, \zeta_2, \zeta_3, \zeta_4, p_i, q_i, \alpha_i, \gamma_i, \gamma_i' \stackrel{\$}{\leftarrow} \mathbb{Z}_q$, for $i \in [n]$, satisfying

$$p_i \alpha_i = \zeta_1 \quad q_i \gamma_i = \zeta_2 \quad q_i \alpha_i = \zeta_3 \quad p_i \gamma_i' = \zeta_4 \quad (5)$$

and output the *secret keys* sk_i and the *encryption keys* ek_i as follows

$$\begin{aligned} \text{sk}_i &:= \left(s_i \alpha_i \mathbf{h}_{i,1}^* + s_i \gamma_i \mathbf{h}_{i,2}^*, u_i \gamma_i' \mathbf{h}_{i,1}^* + u_i \alpha_i \mathbf{h}_{i,2}^*, \frac{v_i}{t_i} \mathbf{h}_{i,3}^* + \mathbf{h}_{i,4}^*, \theta_i \mathbf{h}_{i,6}^* \right) \\ \text{ek}_i &:= (p_i H_i^{(1)} - (\zeta_1 s_i + \zeta_4 u_i) H_i^{(4)}, q_i H_i^{(2)} - (\zeta_2 s_i + \zeta_3 u_i) H_i^{(4)}, \\ &\quad t_i \mathbf{h}_{i,3} - v_i \mathbf{h}_{i,4}, \mathbf{h}_{i,4}, H_i^{(6)}) \end{aligned}$$

where $H_i^{(k)}$ denotes the k -th row of H_i for $i \in [n]$.

DKShare($\text{sk}_i, (\text{tag-f}, \text{info}(\mathbf{y})), y_i$): We assume that the function tag contains tag-f and public information about info(\mathbf{y}). The i -th parameter is $y_i := \mathbf{y}[i]$. Compute $H_2(\text{tag-f}, \text{info}(\mathbf{y})) \rightarrow d_{\text{tag-f}, \mathbf{y}} \in \mathbb{Z}_q$. Parse

$$\text{sk}_i := \left(s_i \alpha_i \mathbf{h}_{i,1}^* + s_i \gamma_i \mathbf{h}_{i,2}^*, u_i \gamma_i' \mathbf{h}_{i,1}^* + u_i \alpha_i \mathbf{h}_{i,2}^*, \frac{v_i}{t_i} \mathbf{h}_{i,3}^* + \mathbf{h}_{i,4}^*, \theta_i \mathbf{h}_{i,6}^* \right) .$$

Sample $z \xleftarrow{\$} \mathbb{Z}_q$ then compute

$$\begin{aligned} \mathbf{k}_{i,\text{ipfe}} &= y_i \cdot (s_i \alpha_i \mathbf{h}_{i,1}^* + s_i \gamma_i \mathbf{h}_{i,2}^*) + y_i \cdot (u_i \gamma_i' \mathbf{h}_{i,1}^* + u_i \alpha_i \mathbf{h}_{i,2}^*) \\ &\quad + y_i \cdot \left(\frac{v_i}{t_i} \mathbf{h}_{i,3}^* + \mathbf{h}_{i,4}^* \right) + d_{\text{tag-f}, \mathbf{y}} \cdot \theta_i \mathbf{h}_{i,6}^* \\ &= \left(s_i y_i \alpha_i + u_i y_i \gamma_i', s_i y_i \gamma_i + u_i y_i \alpha_i, \frac{v_i}{t_i} y_i, y_i, 0, d_{\text{tag-f}, \mathbf{y}} \theta_i \right)_{\mathbf{H}_i^*} . \end{aligned}$$

Output $\text{dk}_{\text{tag-f}, i} := \mathbf{k}_{i,\text{ipfe}}$.

DKeyComb($(\text{dk}_{\text{tag-f}, i})_{i \in [n]}, \text{tag-f}, \mathbf{y}$): Output \perp if there is any incoherence of $d_{\text{tag-f}, \mathbf{y}}$ among the $\text{dk}_{\text{tag-f}, i}$. Otherwise, parse $\text{dk}_{\text{tag-f}, i} := \mathbf{k}_{i,\text{ipfe}}$ and output

$$\text{dk}_{\text{tag-f}, \mathbf{y}} := (\mathbf{k}_{i,\text{ipfe}})_{i \in [n]} .$$

Enc($\text{ek}_i, \text{tag}, x_i$): Parse

$$\begin{aligned} \text{ek}_i &:= (p_i H_i^{(1)} - (\zeta_1 s_i + \zeta_4 u_i) H_i^{(4)}, q_i H_i^{(2)} - (\zeta_2 s_i + \zeta_3 u_i) H_i^{(4)}, \\ &\quad t_i \mathbf{h}_{i,3} - v_i \mathbf{h}_{i,4}, \mathbf{h}_{i,4}, H_i^{(6)}) \end{aligned}$$

and compute $H_1(\text{tag}) \rightarrow ([\omega]_1, [\omega']_1) \in \mathbb{G}_1^2$ and sample $r_i \xleftarrow{\$} \mathbb{Z}_q$. Compute

$$\begin{aligned} \mathbf{c}_{i,\text{ipfe}} &= (p_i H_i^{(1)} - (\zeta_1 s_i + \zeta_4 u_i) H_i^{(4)}) \cdot [\omega]_1 + (q_i H_i^{(2)} - (\zeta_2 s_i + \zeta_3 u_i) H_i^{(4)}) \cdot [\omega']_1 \\ &\quad + r_i \cdot (t_i \mathbf{h}_{i,3} - v_i \mathbf{h}_{i,4}) + x_i \mathbf{h}_{i,4} + H_i^{(6)} [\omega]_1 \\ &= (\omega p_i, \omega' q_i, r_i t_i, -(\omega \zeta_1 + \omega' \zeta_2) \cdot s_i - (\omega' \zeta_3 + \omega \zeta_4) \cdot u_i + x_i - r_i v_i, 0, \omega)_{\mathbf{H}_i} . \end{aligned}$$

and output $\text{ct}_{\text{tag}, i} := \mathbf{c}_{i,\text{ipfe}}$.

Dec($\text{dk}_{\text{tag-f}, \mathbf{y}}, \mathbf{c}$): Parse $\text{dk}_{\text{tag-f}, \mathbf{y}} = (\mathbf{k}_{i,\text{ipfe}})_{i \in [n]}$ and $\mathbf{c} := (\text{ct}_{\text{tag}, i})_i$. Finally, compute the discrete logarithm in base $>$ of $[\text{out}]_t = \sum_{i=1}^n (\text{ct}_{\text{tag}, i} \times \mathbf{k}_{i,\text{ipfe}})$ and output the small value out.

The *correctness* of the scheme is verified by:

$$\begin{aligned} &[\text{out}]_t \\ &= \sum_{i=1}^n (\text{ct}_{\text{tag}, i} \times \mathbf{k}_{i,\text{ipfe}}) \\ &= \sum_{i=1}^n \left(\begin{array}{c} (\omega p_i, \omega' q_i, r_i t_i, -(\omega \zeta_1 + \omega' \zeta_2) \cdot s_i - (\omega' \zeta_3 + \omega \zeta_4) \cdot u_i + x_i - r_i v_i, \\ 0, \omega)_{\mathbf{H}_i} \\ \times \\ (s_i y_i \alpha_i + u_i y_i \gamma_i', s_i y_i \gamma_i + u_i y_i \alpha_i, \frac{v_i}{t_i} y_i, y_i, \\ 0, d_{\text{tag-f}, \mathbf{y}} \theta_i)_{\mathbf{H}_i^*} \end{array} \right) \end{aligned}$$

$$\begin{aligned}
 &\stackrel{(*)}{=} \sum_{i=1}^n \llbracket \omega \zeta_1 s_i y_i + \omega \zeta_4 u_i y_i + \omega' \zeta_2 s_i y_i + \omega' \zeta_3 u_i y_i + \theta_i d_{\text{tag-f,y}\omega} \rrbracket_t \\
 &\quad + \sum_{i=1}^n \llbracket (-(\omega \zeta_1 + \omega' \zeta_2) \cdot s_i - (\omega' \zeta_3 + \omega \zeta_4) \cdot u_i + x_i) \cdot y_i \rrbracket_t \\
 &= \llbracket (\omega \zeta_1 + \omega' \zeta_2) \cdot \langle S, \mathbf{y} \rangle + (\omega' \zeta_3 + \omega \zeta_4) \cdot \langle U, \mathbf{y} \rangle \rrbracket_t + \sum_{i=1}^n \llbracket \theta_i d_{\text{tag-f,y}\omega} \rrbracket_t \\
 &\quad - \llbracket (\omega \zeta_1 + \omega' \zeta_2) \cdot \langle S, \mathbf{y} \rangle + (\omega' \zeta_3 + \omega \zeta_4) \cdot \langle U, \mathbf{y} \rangle \rrbracket_t + \llbracket \langle \mathbf{x}, \mathbf{y} \rangle \rrbracket_t \\
 &= \llbracket \langle \mathbf{x}, \mathbf{y} \rangle \rrbracket_t,
 \end{aligned}$$

where the equality (*) comes from system (5). We recall that $(\theta_i)_{i \in [n]}$ is a secret sharing of 0.

5.2 Adaptive Security Against Static Corruptions of Secret Keys

We now give the security theorem of one time IND-security for our construction from Sect. 5.1, *adaptively* in the challenge messages with *dynamic* corruption of encryption keys and *static* corruption of secret keys. We refer to Remark 15 for the concrete interpretation of the security model. The full proof can be found in the full version [25].

Theorem 16. *Let $\mathcal{E} = (\text{Setup}, \text{DKShare}, \text{DKeyComb}, \text{Enc}, \text{Dec})$ be a DMCFE candidate for \mathcal{F}^{IP} from Sect. 5.1 in a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, \cdot, \langle \cdot, \cdot \rangle, e, q)$. Then, \mathcal{E} is IND-secure with static corruption of secret keys in the ROM if the SXDH assumption holds for \mathbb{G}_1 and \mathbb{G}_2 . More specifically, let n denote the dimension for inner-products, K denote the maximum number of key queries, and Q_1, Q_2 denote the maximum number of random oracle (RO) queries to $\mathbb{H}_1, \mathbb{H}_2$ respectively. For any ppt adversary \mathcal{A} against \mathcal{E} with static secret key corruption and one-time challenge, we have the following bound:*

$$\text{Adv}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{dmc-stat-sk-ind-cpa-1chal}}(1^\lambda) \leq (3 + 2Q_1 + K) \cdot \text{Adv}_{\mathbb{G}_1, \mathbb{G}_2}^{\text{SXDH}}(1^\lambda) + \frac{Q_2^2}{2q}$$

and in the reduction there is an additive loss $\mathcal{O}(Q_1 \cdot t_{\mathbb{G}_1} + Q_2 \cdot t_{\mathbb{G}_2})$ in time, where $t_{\mathbb{G}_1}, t_{\mathbb{G}_2}$ is the cost for one addition in $\mathbb{G}_1, \mathbb{G}_2$.

6 DMCFE for Inner-Products with Stronger Security Against Incomplete Queries

In this section, we show how to obtain a DMCFE scheme that is IND-secure against chosen plaintext attacks without *complete* queries restriction (see condition 2 in Sect. 3.1), under our stronger admissibility notion. The definition of security notion for the new setting is restated so that admissible adversaries can query *incomplete* challenge ciphertexts as well as *incomplete* functional keys.

Definition 17 (Admissible attacks with incomplete queries). Let \mathcal{A} be a ppt adversary and let

$$\mathcal{E} = (\text{Setup}, \text{DKShare}, \text{DKeyComb}, \text{Enc}, \text{Dec})$$

be a DMCFE scheme for a function class \mathcal{F} set up w.r.t $\lambda \in \mathbb{N}$. We denote by $\text{rand}_{\text{Chall}}$ the random coins of the challenger and $\text{rand}_{\mathcal{A}}$ the random coins of the adversary in an experiment given in Fig. 1. In **Finalize**, considering the queries $(\mathcal{Q}, \mathcal{Q}_{\text{Enc}}, \mathcal{C}_{\text{skey}}, \mathcal{C}_{\text{ekey}}, \{(\mathbf{x}_0^*, \mathbf{x}_1^*, \text{tag})\}, \{(\mathbf{x}, \text{tag}^{(k)})\})$, we say that the attack corresponding to these queries is NOT admissible if the following is satisfied

There exist $\text{tag}, \text{tag-f} \in \text{Tag}$, a function $F \in \mathcal{F}$, two challenges $(x_i^{(0)}, x_i^{(1)})_{i \in [n]}$ such that $(F, \text{tag-f})$ is queried to **DKeyGenShare** for all honest components, $((x_i^{(0)}, x_i^{(1)})_{i \in [n]}, \text{tag})$ is queried to **LoR** for all honest components, and there exists a pair $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)})$ deducible from $(\mathbf{x}_{\text{ekey}}^{(0)}, \mathbf{x}_{\text{ekey}}^{(1)})$, a function G deducible from \mathbf{y}_{skey} satisfying

$$G(\mathbf{z}^{(0)}) \neq G(\mathbf{z}^{(1)}) ,$$

where we define $\mathbf{y}_{\text{skey}} := (y_i)_{i \in \mathcal{H}_{\text{skey}}}$ and for $b \in \{0, 1\}$, $\mathbf{x}_{\text{ekey}}^{(b)} := (x_i^{(b)})_{i \in \mathcal{H}_{\text{ekey}}}$.

Otherwise, we say that the attack is admissible.

Definition 18 (IND+-security for DMCFE). A DMCFE scheme

$$\mathcal{E} = (\text{Setup}, \text{DKShare}, \text{DKeyComb}, \text{Enc}, \text{Dec})$$

for the function class $\mathcal{F} = \{F_\lambda\}_{\lambda \in \mathbb{N}}$ is IND+-secure if for all ppt adversaries \mathcal{A} , and for all sufficiently large $\lambda \in \mathbb{N}$, the following probability is negligible

$$\text{Adv}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{xx}+}(1^\lambda) := \left| \Pr[\text{Expr}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{xx}+}(1^\lambda) = 1] - \frac{1}{2} \right| .$$

The probability is taken over coins of \mathcal{A} and the algorithms. The indicator xx can be among $\{\text{dmc-ind-cpa}, \text{dmc-sel-ind-cpa}, \text{dmc-stat-ind-cpa}, \text{dmc-ind-cpa-1chal}\}$ ⁶. The experiment $\text{Expr}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{xx}+}(1^\lambda)$ is the same as $\text{Expr}_{\mathcal{E}, \mathcal{F}, \mathcal{A}}^{\text{xx}}$ depicted in Fig. 1, except that we use Definition 17 for the admissibility condition in **Finalize**.

6.1 Constructions

Generic Transformation with Security Against Selective Challenges.

We follow the same method in [17] and apply generically a layer of using a primitive called *All-or-Nothing Encapsulation (AoNE)*, so as to make our scheme from Sect. 5.1 secure in our *stronger* security model against incomplete queries. Our AoNE-based transformation uses the generic AoNE from [17], which in turn is

⁶ Similarly, we can allow dynamic corruption on one type but static corruption on the other type of keys, such as *dmc-stat-sk-ind-cpa+* to indicate *partially static IND-security* with adaptive challenges, dynamic corruption of *ekey*, and static corruption of *skey*.

built on top of a *one-time secure symmetric encryption* (OT-SE). In the security proof, which can be naturally adapted from [17, Theorem 26], this OT-SE prevents programing conveniently to achieve adaptive security w.r.t the challenge ciphertxts. We also remark that the static corruption is unavoidable since the security of AoNE makes sense only when being applied on honest components, for the security reduction. This generic transformation is provably secure under *static* corruption and *selective* challenges. The transformation is presented in the full version [25].

Concrete Scheme with Security Against Adaptive Challenges. We present a concrete adaptation of our base DMCFE scheme from Sect. 5.1 to satisfy the stronger security notion against *incomplete* challenge ciphertxts as well as *incomplete* functional keys, with minimal modifications being put in boxed components for the ease of comparison. The function class stays the same as in Sect. 5.1, for which our admissibility is optimal, see the full version [25]. In contrast to the generic transformation, we build the AoNE concretely by combining one-time pad (OTP) and a random oracle (RO). Then, the programmability of the RO helps us circumvent the problem of adaptive queries. While programming the RO, we indeed exploits in a non-blackbox manner the OTP as a summation in \mathbb{Z}_q^* to accumulate a secret sharing of 0 on the honest parts (known in advance thanks to static corruption).

The details of our construction go as follows:

Setup(1^λ): Sample two full-domain hash functions $H_1 : \text{Tag} \rightarrow \mathbb{G}_1^2$ and $H_2 : \text{Tag} \times \mathbb{Z}_q^n \rightarrow \mathbb{G}_2$. Choose n pairs of dual orthogonal bases $(\mathbf{H}_i, \mathbf{H}_i^*)$ for $i \in [n]$, where $(\mathbf{H}_i, \mathbf{H}_i^*)$ is a pair of dual bases for $(\mathbb{G}_1^8, \mathbb{G}_2^8)$. We denote the basis changing matrices for $(\mathbf{H}, \mathbf{H}^*)$ as (H_i, H'_i) :

$$(\mathbf{H}_i = H_i \cdot \mathbf{T}; \mathbf{H}_i^* = H'_i \cdot \mathbf{T}^*)_{i \in [n]}$$

where $H_i, H'_i \in \mathbb{Z}_q^{8 \times 8}$ and $(\mathbf{T} = \llbracket I_8 \rrbracket_1, \mathbf{T}^* = \llbracket I_8 \rrbracket_2)$ are canonical bases of $(\mathbb{G}_1^8, \mathbb{G}_2^8)$, for the identity matrix I_8 . We recall that interactions are involved only in this Setup phase. For each $i \in [n]$, we write

$$\mathbf{H}_i = (\mathbf{h}_{i,1}, \mathbf{h}_{i,2}, \dots, \mathbf{h}_{i,8}) \quad \mathbf{H}_i^* = (\mathbf{h}_{i,1}^*, \mathbf{h}_{i,2}^*, \dots, \mathbf{h}_{i,8}^*)$$

and sample $\zeta_1, \zeta_2, \zeta_3, \zeta_4, S, U, V, T, \llbracket D, E \rrbracket \stackrel{\$}{\leftarrow} (\mathbb{Z}_q^*)^n$ where $S = (s_1, \dots, s_n)$, $U = (u_1, \dots, u_n)$, $V = (v_1, \dots, v_n)$, $T = (t_1, \dots, t_n)$, $D = (d_1, \dots, d_n)$, and $E = (e_1, \dots, e_n)$. Sample $\theta_1, \dots, \theta_n \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$ such that $\sum_{i=1}^n \theta_i = 0$ and for $i \in [n]$ let $p_i, q_i, \alpha_i, \gamma_i, \gamma'_i \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ satisfy

$$p_i \alpha_i = \zeta_1 \quad q_i \gamma_i = \zeta_2 \quad q_i \alpha_i = \zeta_3 \quad p_i \gamma'_i = \zeta_4$$

We set the public parameters to be $(\llbracket \langle E, \mathbf{1} \rangle \rrbracket_1, \llbracket \langle D, \mathbf{1} \rangle \rrbracket_2)$. Sample $\llbracket \epsilon, \delta \rrbracket \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ and generate random n -out-of- n secret sharings $\llbracket (\epsilon_i)_i, (\delta_i)_i \rrbracket$ of ϵ, δ so that

$\sum_{i=1}^n \epsilon_i = \epsilon$, $\sum_{i=1}^n \delta_i = \delta$. Output the *secret keys* \mathbf{sk}_i and the *encryption keys* \mathbf{ek}_i as follows

$$\begin{aligned} \mathbf{sk}_i &:= (\boxed{\epsilon_i}, s_i \alpha_i \mathbf{h}_{i,1}^* + s_i \gamma_i \mathbf{h}_{i,2}^*, u_i \gamma_i' \mathbf{h}_{i,1}^* + u_i \alpha_i \mathbf{h}_{i,2}^*, -\frac{v_i}{t_i} \mathbf{h}_{i,3}^* + \mathbf{h}_{i,4}^*, \\ &\quad \boxed{\theta_i H_i^{(6)} - e_i H_i^{(8)}, \epsilon \mathbf{h}_{i,8}}) \\ \mathbf{ek}_i &:= (\boxed{\delta_i}, p_i H_i^{(1)} - (\zeta_1 s_i + \zeta_4 u_i) H_i^{(4)} - \boxed{d_i H_i^{(7)}}, q_i H_i^{(2)} - (\zeta_2 s_i + \zeta_3 u_i) H_i^{(4)}, \\ &\quad t_i \mathbf{h}_{i,3} - v_i \mathbf{h}_{i,4}, \mathbf{h}_{i,4}, H_i^{(6)}, \boxed{\delta \mathbf{h}_{i,7}^*}) \end{aligned}$$

where $H_i^{(k)}$ denotes the k -th row of H_i for $i \in [n]$ and $\mathbf{1} = (1, \dots, 1)$. $\text{DKShare}(\mathbf{sk}_i, (\text{tag-f}, \text{info}(\mathbf{y})), y_i)$: We assume that the function tag contains tag-f and public information about info(\mathbf{y}). The i -th parameter is $y_i := \mathbf{y}[i]$. We will use a full-domain hash function $\text{H}_2 : \text{Tag} \times \mathbb{Z}_q^n \rightarrow \mathbb{G}_2$. Parse

$$\begin{aligned} \mathbf{sk}_i &:= (\epsilon_i, s_i \alpha_i \mathbf{h}_{i,1}^* + s_i \gamma_i \mathbf{h}_{i,2}^*, u_i \gamma_i' \mathbf{h}_{i,1}^* + u_i \alpha_i \mathbf{h}_{i,2}^*, \frac{v_i}{t_i} \mathbf{h}_{i,3}^* + \mathbf{h}_{i,4}^*, \\ &\quad \theta_i H_i^{(6)} - e_i H_i^{(8)}, \epsilon \mathbf{h}_{i,8}) . \end{aligned}$$

Compute $\text{H}_2(\text{tag-f}, \text{info}(\mathbf{y})) \rightarrow \llbracket \kappa_{\text{tag-f}, \mathbf{y}} \rrbracket_2$ and

$$\begin{aligned} &\mathbf{k}_{i, \text{ipfe}} \\ &= y_i \cdot (s_i \alpha_i \mathbf{h}_{i,1}^* + s_i \gamma_i \mathbf{h}_{i,2}^*) + y_i \cdot (u_i \gamma_i' \mathbf{h}_{i,1}^* + u_i \alpha_i \mathbf{h}_{i,2}^*) \\ &\quad + y_i (\frac{v_i}{t_i} \mathbf{h}_{i,3}^* + \mathbf{h}_{i,4}^*) + (\theta_i H_i^{(6)} - e_i H_i^{(8)}) \cdot \llbracket \kappa_{\text{tag-f}, \mathbf{y}} \rrbracket_2 \\ &= (s_i y_i \alpha_i + u_i y_i \gamma_i', s_i y_i \gamma_i + u_i y_i \alpha_i, \frac{v_i}{t_i} y_i, y_i, 0, \kappa_{\text{tag-f}, \mathbf{y}} \theta_i, \boxed{0, -e_i \kappa_{\text{tag-f}, \mathbf{y}}})_{\mathbf{H}_i^*} \\ &\quad \boxed{\epsilon(\epsilon_i \cdot \llbracket \langle E, \mathbf{1} \rangle \rrbracket_1, \llbracket \kappa_{\text{tag-f}, \mathbf{y}} \rrbracket_2)} = \llbracket \epsilon_i \langle E, \mathbf{1} \rangle \kappa_{\text{tag-f}, \mathbf{y}} \rrbracket_{\mathbf{t}} \end{aligned}$$

where $\llbracket \langle E, \mathbf{1} \rangle \rrbracket_1$ is public. Output $\mathbf{dk}_{\text{tag-f}, i} := (\mathbf{k}_{i, \text{ipfe}}, \boxed{\epsilon \cdot \mathbf{h}_{i,8}, \llbracket \epsilon_i \langle E, \mathbf{1} \rangle \kappa_{\text{tag-f}, \mathbf{y}} \rrbracket_{\mathbf{t}}})$. $\text{DKeyComb}(\mathbf{dk}_{\text{tag-f}, i}, \text{tag-f}, \mathbf{y})$: Output \perp if there is any incoherence among the $\mathbf{dk}_{\text{tag-f}, i}$. Else, let $\mathbf{dk}_{\text{tag-f}, i} := (\mathbf{k}_{i, \text{ipfe}}, \epsilon \cdot \mathbf{h}_{i,8}, \llbracket \epsilon_i \langle E, \mathbf{1} \rangle \kappa_{\text{tag-f}, \mathbf{y}} \rrbracket_{\mathbf{t}})$.

Compute $\llbracket \epsilon \langle E, \mathbf{1} \rangle \kappa_{\text{tag-f}, \mathbf{y}} \rrbracket_{\mathbf{t}} = \sum_{i=1}^n \llbracket \epsilon_i \langle E, \mathbf{1} \rangle \kappa_{\text{tag-f}, \mathbf{y}} \rrbracket_{\mathbf{t}}$ and output

$$\mathbf{dk}_{\text{tag-f}, \mathbf{y}} := ((\mathbf{k}_{i, \text{ipfe}}, \boxed{\epsilon \cdot \mathbf{h}_{i,8}})_{i \in [n]}, \llbracket \epsilon \langle E, \mathbf{1} \rangle \kappa_{\text{tag-f}, \mathbf{y}} \rrbracket_{\mathbf{t}}) .$$

$\text{Enc}(\mathbf{ek}_i, \text{tag}, x_i)$: Parse

$$\begin{aligned} \mathbf{ek}_i &:= (\delta_i, p_i H_i^{(1)} - (\zeta_1 s_i + \zeta_4 u_i) H_i^{(4)} - d_i H_i^{(7)}, q_i H_i^{(2)} - (\zeta_2 s_i + \zeta_3 u_i) H_i^{(4)}, \\ &\quad t_i \mathbf{h}_{i,3} - v_i \mathbf{h}_{i,4}, \mathbf{h}_{i,4}, H_i^{(6)}, \delta \cdot \mathbf{h}_{i,7}^*) \end{aligned}$$

and compute $\text{H}_1(\text{tag}) \rightarrow (\llbracket \omega \rrbracket_1, \llbracket \omega' \rrbracket_1) \in \mathbb{G}_1^2$; and sample $r_i \xleftarrow{\$} \mathbb{Z}_q$. Compute

$$\begin{aligned} &\mathbf{c}_{i, \text{ipfe}} \\ &= (p_i H_i^{(1)} - (\zeta_1 s_i + \zeta_4 u_i) H_i^{(4)} - d_i H_i^{(7)}) \cdot \llbracket \omega \rrbracket_1 + (q_i H_i^{(2)} - (\zeta_2 s_i + \zeta_3 u_i) H_i^{(4)}) \cdot \\ &\quad \llbracket \omega' \rrbracket_1 \\ &\quad + r_i \cdot (t_i \mathbf{h}_{i,3} - v_i \mathbf{h}_{i,4}) + x_i \mathbf{h}_{i,4} + H_i^{(6)} \llbracket \omega \rrbracket_1 \end{aligned}$$

$$= (\omega p_i, \omega' q_i, r_i t_i, -(\omega \zeta_1 + \omega' \zeta_2) \cdot s_i - (\omega' \zeta_3 + \omega \zeta_4) \cdot u_i + x_i - r_i v_i, 0, \omega, \boxed{-d_i \omega, 0})_{\mathbf{H}_i}$$

$$\mathbf{e}(\llbracket \omega \rrbracket_2, \delta_i \cdot \llbracket \langle D, \mathbf{1} \rangle \rrbracket_2) = \llbracket \delta_i \langle D, \mathbf{1} \rangle \omega \rrbracket_t$$

where $\llbracket \langle D, \mathbf{1} \rangle \rrbracket_2$ comes from the public parameters.

Output $\mathbf{ct}_{\text{tag},i} := (\mathbf{c}_{i,\text{ipfe}}, \boxed{\delta \cdot \mathbf{h}_{i,7}^*, \llbracket \delta_i \langle D, \mathbf{1} \rangle \omega \rrbracket_t})$.

Dec($\mathbf{dk}_{\text{tag-f},\mathbf{y}}, \mathbf{c}$): Parse

$$\mathbf{dk}_{\text{tag-f},\mathbf{y}} = ((\mathbf{k}_{i,\text{ipfe}}, \boxed{\epsilon \cdot \mathbf{h}_{i,8}})_i, \llbracket \epsilon \langle E, \mathbf{1} \rangle \kappa_{\text{tag-f},\mathbf{y}} \rrbracket_t);$$

$$\mathbf{c} = (\mathbf{c}_{i,\text{ipfe}}, \boxed{\delta \cdot \mathbf{h}_{i,7}^*, \llbracket \delta_i \langle D, \mathbf{1} \rangle \omega \rrbracket_t})_{i=1}^n$$

Compute $\llbracket \delta \langle D, \mathbf{1} \rangle \omega \rrbracket_t = \sum_{i=1}^n \llbracket \delta_i \langle D, \mathbf{1} \rangle \omega \rrbracket_t$ and

$$\llbracket \text{out} \rrbracket_t = \sum_{i=1}^n \left((\mathbf{ct}_{\text{tag},i} + \boxed{\epsilon \cdot \mathbf{h}_{i,8}}) \times (\mathbf{k}_{i,\text{ipfe}} + \boxed{\delta \cdot \mathbf{h}_{i,7}^*}) \right) + \llbracket \epsilon \langle E, \mathbf{1} \rangle \kappa_{\text{tag-f},\mathbf{y}} \rrbracket_t + \llbracket \delta \langle D, \mathbf{1} \rangle \omega \rrbracket_t.$$

Finally, compute the discrete logarithm and output the small value out .

The *correctness* of the scheme is verified by:

$$\begin{aligned} & \llbracket \text{out} \rrbracket_t \\ &= \sum_{i=1}^n \left((\mathbf{k}_{i,\text{ipfe}} + \delta \cdot \mathbf{h}_{i,7}^*) \times (\mathbf{ct}_{\text{tag},i} + \epsilon \cdot \mathbf{h}_{i,8}) \right) + \llbracket \epsilon \langle E, \mathbf{1} \rangle \kappa_{\text{tag-f},\mathbf{y}} \rrbracket_t + \llbracket \delta \langle D, \mathbf{1} \rangle \omega \rrbracket_t \\ &= \sum_{i=1}^n \left(\begin{array}{c} (s_i y_i \alpha_i + u_i y_i \gamma'_i, s_i y_i \gamma_i + u_i y_i \alpha_i, \frac{v_i}{t_i} y_i, y_i, \\ 0, \kappa_{\text{tag-f},\mathbf{y}} \theta_i, \delta, -e_i \kappa_{\text{tag-f},\mathbf{y}})_{\mathbf{H}_i^*} \\ \times \\ (\omega p_i, \omega' q_i, r_i t_i, -(\omega \zeta_1 + \omega' \zeta_2) \cdot s_i - (\omega' \zeta_3 + \omega \zeta_4) \cdot u_i + x_i - r_i v_i, \\ 0, \omega, -d_i \omega, \epsilon)_{\mathbf{H}_i} \end{array} \right) \\ & \quad + \llbracket \epsilon \langle E, \mathbf{1} \rangle \kappa_{\text{tag-f},\mathbf{y}} \rrbracket_t + \llbracket \delta \langle D, \mathbf{1} \rangle \omega \rrbracket_t \\ & \stackrel{(*)}{=} \sum_{i=1}^n \llbracket \omega \zeta_1 s_i y_i + \omega \zeta_4 u_i y_i + \omega' \zeta_2 s_i y_i + \omega' \zeta_3 u_i y_i + \theta_i \omega \kappa_{\text{tag-f},\mathbf{y}} \rrbracket_t \\ & \quad + \sum_{i=1}^n \llbracket (-(\omega \zeta_1 + \omega' \zeta_2) \cdot s_i - (\omega' \zeta_3 + \omega \zeta_4) \cdot u_i + x_i) \cdot y_i \rrbracket_t \\ & \quad + \sum_{i=1}^n \llbracket (-(\omega \zeta_1 + \omega' \zeta_2) \cdot s_i - (\omega' \zeta_3 + \omega \zeta_4) \cdot u_i + x_i) \cdot y_i \rrbracket_t \\ & = \llbracket \langle \mathbf{x}, \mathbf{y} \rangle \rrbracket_t . \end{aligned}$$

where the equality $(*)$ comes from system (5). We recall that $(\theta_i)_{i \in [n]}$ is a secret sharing of 0.

Security. We prove the one-time static security of our DMCFE scheme in the ROM, where the full-domain hash functions are modeled as random oracles, the sets of corrupted clients $\mathcal{C}_{\text{ekey}}$ as well as $\mathcal{C}_{\text{skey}}$ must be sent up front (*static*

corruption), while the challenges $(\mathbf{x}_0^*, \mathbf{x}_1^*)$ can be adaptively chosen (*adaptive challenge*). We note that we can achieve a better level of security in our concrete instantiation compared to the generic transformation. On one hand, our transformation follows the same blueprint in the work by Chotard *et al.* [17], which is the most relevant to our DMCFE setting. We apply a layer of *All-or-Nothing Encapsulation* (AoNE) to our ciphertext and key components, which ensures that the original key/ciphertext components can be recovered only when all parts are gathered. Our concrete DMCFE in Sect. 6 builds the AoNE directly by combining one-time pad (OTP) and a random oracle (RO). Then, the programmability of the RO helps us circumvent the problem of adaptive queries. While programming the RO, we indeed exploits in a non-blackbox manner the OTP as a summation in \mathbb{Z}_q^* to accumulate a secret sharing of 0 on the honest parts (known in advance thanks to static corruption).

Theorem 19 *Let $\mathcal{E} = (\text{Setup}, \text{DKShare}, \text{DKeyComb}, \text{Enc}, \text{Dec})$ be the DMCFE constructed in Sect. 6.1. Then, \mathcal{E} is one-time statically IND+-secure in the ROM following the security model in Definition 18 if the SXDH and DBDH assumptions hold for \mathbb{G}_1 and \mathbb{G}_2 . More specifically, let n denote the dimension for inner-products, Q_1, Q_2 denote the maximum number of random oracle (RO) queries to H_1, H_2 and K denote the total number of functional key queries. For any one-time challenge ppt adversary \mathcal{A} against \mathcal{E} with static corruption of secret keys and encryption keys, we have the following bound:*

$$\text{Adv}_{\mathcal{E}, \mathcal{F}^{\text{IP}}, \mathcal{A}}^{\text{dmc-stat-1chal}^+}(1^\lambda) \leq (K + 1)\text{Adv}_{\mathbb{G}_1, \mathbb{G}_2}^{\text{DBDH}}(1^\lambda) + (3 + 2Q_1 + K)\text{Adv}_{\mathbb{G}_1, \mathbb{G}_2}^{\text{SXDH}}(1^\lambda) + \frac{Q_2^2}{2q}.$$

Details are presented in the full version [25].

Acknowledgment. This work was supported in part by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute, the French ANR Project ANR-19-CE39-0011 PRESTO and the Beyond5G Project as part of the plan “France Relance”.

References

1. Abdalla, M., Bellare, M., Neven, G.: Robust encryption. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 480–497. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11799-2_28
2. Abdalla, M., Benhamouda, F., Gay, R.: From single-input to multi-client inner-product functional encryption. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 552–582. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_19
3. Abdalla, M., Benhamouda, F., Kohlweiss, M., Waldner, H.: Decentralizing inner-product functional encryption. In: Lin, D., Sako, K. (eds.) PKC 2019. LNCS, vol. 11443, pp. 128–157. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17259-6_5
4. Abdalla, M., Gay, R., Raykova, M., Wee, H.: Multi-input inner-product functional encryption from pairings. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 601–626. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_21

5. Agrawal, S., Bhattacharjee, S., Phan, D.H., Stehlé, D., Yamada, S.: Efficient public trace and revoke from standard assumptions: Extended abstract. In: ACM CCS 2017 (2017)
6. Agrawal, S., Goyal, R., Tomida, J.: Multi-input quadratic functional encryption from pairings. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 208–238. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_8
7. Agrawal, S., Goyal, R., Tomida, J.: Multi-party functional encryption. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13043, pp. 224–255. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90453-1_8
8. Agrawal, S., Goyal, R., Tomida, J.: Multi-input quadratic functional encryption: Stronger security, broader functionality. In: Kiltz, E., Vaikuntanathan, V. (eds.) Theory of Cryptography. TCC 2022. LNCS, vol. 13747. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22318-1_25
9. Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 333–362. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_12
10. Bellare, M., O’Neill, A.: Semantically-secure functional encryption: possibility results, impossibility results and the quest for a general definition. In: CANS 13 (2013)
11. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_13
12. Boneh, D., Gentry, C., Hamburg, M.: Space-efficient identity based encryption without pairings. In: 48th FOCS (2007)
13. Boneh, D., Sahai, A., Waters, B.: Functional encryption: definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_16
14. Chen, J., Lim, H.W., Ling, S., Wang, H., Wee, H.: Shorter IBE and signatures via asymmetric pairings. In: PAIRING 2012 (2013)
15. Chotard, J., Dufour Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Decentralized multi-client functional encryption for inner product. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 703–732. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_24
16. Chotard, J., Dufour Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Multi-client functional encryption with repetition for inner product. Cryptology ePrint Archive, Report 2018/1021 (2018). <https://eprint.iacr.org/2018/1021>
17. Chotard, J., Dufour-Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Dynamic decentralized functional encryption. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12170, pp. 747–775. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56784-2_25
18. Cocks, C.: An identity based encryption scheme based on quadratic residues. In: 8th IMA International Conference on Cryptography and Coding (2001)
19. Datta, P., Okamoto, T., Tomida, J.: Full-hiding (unbounded) multi-input inner product functional encryption from the k -Linear assumption. In: PKC 2018, Part II (2018)
20. Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for Diffie-Hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 129–147. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_8

21. Goldwasser, S., et al.: Multi-input functional encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 578–602. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_32
22. Gordon, S.D., Katz, J., Liu, F.H., Shi, E., Zhou, H.S.: Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/774 (2013), <https://eprint.iacr.org/2013/774>
23. Lewko, A., Waters, B.: New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 455–479. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11799-2_27
24. Libert, B., Tîţiu, R.: Multi-client functional encryption for linear functions in the standard model from LWE. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 520–551. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_18
25. Nguyen, K., Phan, D.H., Pointcheval, D.: Optimal security notion for decentralized multi-client functional encryption. In: 21st International Conference on Applied Cryptography and Network Security. Springer-Verlag (2023). <https://eprint.iacr.org/2023/435>
26. Okamoto, T., Takashima, K.: Fully secure functional encryption with general relations from the decisional linear assumption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 191–208. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_11
27. Okamoto, T., Takashima, K.: Adaptively attribute-hiding (hierarchical) inner product encryption. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 591–608. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_35
28. Okamoto, T., Takashima, K.: Fully secure unbounded inner-product and attribute-based encryption. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 349–366. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_22
29. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_27
30. Shamir, A.: Identity-based cryptosystems and signature schemes. In: CRYPTO1984 (1984)
31. Waters, B.: Dual system encryption: realizing fully secure IBE and HIBE under simple assumptions. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 619–636. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_36



Anonymous (Hierarchical) Identity-Based Encryption from Broader Assumptions

Huangting Wu and Sherman S. M. Chow^(✉) 

Department of Information Engineering, The Chinese University of Hong Kong,
Shatin, N.T., Hong Kong
{wh017, smchow}@ie.cuhk.edu.hk

Abstract. Döttling and Garg (CRYPTO 2017) introduced a non-black-box approach to identity-based encryption (IBE). This paves the way for the first and still the only anonymous IBE (AIBE) scheme from the computational Diffie–Hellman (CDH) assumption of Brakerski *et al.* (EUROCRYPT 2018). This paper revisits the blinding technique of Brakerski *et al.* and introduces a suite of blind primitives, extending chameleon encryption, hash encryption, and one-time signature with encryption. Using them, we propose an AIBE scheme from CDH with improved efficiency compared to Brakerski *et al.*, especially in the decryption time. We also propose the first anonymous hierarchical IBE (AHIBE) scheme from CDH and the first AIBE and AHIBE schemes from the ϕ -hiding assumption, with similar efficiency as their non-anonymous counterparts.

Keywords: identity-based encryption · hierarchical IBE · anonymity · computational Diffie–Hellman · ϕ -hiding

1 Introduction

Identity-based encryption (IBE) [5] allows a sender to encrypt messages to a receiver without knowing the receiver-specific public key, but only using the receiver’s identity and a master public key that is small, i.e., polynomial in the security parameter. For decryption, the receiver is issued a secret key corresponding to its identity. Hierarchical IBE (HIBE) [15] further supports key delegation. The secret key or the delegate key of identity id enables the generation of secret/delegate keys for any identities with prefix id . HIBE found applications in forward-secure public-key encryption [10] and timed-released encryption [14]. Anonymous IBE (AIBE) [4] and anonymous HIBE (AHIBE) [7] further hide the recipient identity of ciphertexts from unauthorized decryptors. Anonymity can be leveraged to defend against the inherent key escrow of IBE [13]. A(H)IBE found applications in public-key (and identity-based) searchable encryption [1].

Most (H)IBE schemes are constructed from pairing [3, 5, 21, 27, 28] or lattice [2, 11] assumptions. It was unclear whether the computational Diffie–

Sherman Chow is supported in part by the General Research Funds (CUHK 14210621 and 14209918), University Grants Committee, Hong Kong.

Hellman (CDH) assumption suffices until the result of Döttling and Garg [16, 17]. They introduce a non-black-box approach to construct (H)IBE from CDH. Non-black-box approaches to encryption typically defer parts of encryption procedures to decryptors, which prevents encryptors from learning the otherwise required inputs fed to those partial encryption procedures. Such approaches bypass the black-box impossibility results of IBE [6, 26]. Its core building block is an encryption primitive called chameleon encryption (CE), which associates a chameleon hash function with an encryption scheme. Chameleon hash supports collision tractability – one can use the trapdoor to efficiently find a collision (i.e., another pre-image) of a given hash value by outputting another chameleon randomness.

This approach then enjoys many developments, starting with generic constructions of (H)IBE from a new notion named one-time signatures with encryption (OTSE) [15]. Döttling *et al.* [18] then proposed a generic OTSE construction from hash encryption (HE), a simplification of chameleon encryption. Garg and Hajiabadi [19] introduced one-way function with encryption (OWFE), a close relative to CE, HE, and OTSE. Goyal *et al.* [24] provided a new OWFE scheme from the ϕ -hiding assumption. All these primitives lead to many interesting cryptographic results, e.g., laconic oblivious transfer [12], registration-based encryption [20], trapdoor functions [19], and chosen-ciphertext security [25].

Inspired by the non-black-box approach, Brakerski *et al.* [8] proposed the first and still the only AIBE scheme from CDH. Somewhat surprisingly, despite the above two lines of development, we see not many developments of non-black-box anonymous IBE. We are intrigued to ask:

Can we extend the applicability of Brakerski et al.'s blinding technique [8]? Specifically, can we push the frontier of anonymous IBE in terms of efficiency, key delegation, or the spectrum of intractability assumptions?

1.1 Our Results

In this work, we propose an AIBE scheme from CDH, which greatly improves the efficiency of the AIBE scheme of Brakerski *et al.* [8], especially in terms of decryption time. We also propose the first AHIBE scheme from CDH and the first AIBE and AHIBE schemes from the ϕ -hiding assumption [9], which was never used to construct (H)IBE explicitly before. The former result is only slightly less efficient compared to the existing non-anonymous hierarchical IBE (HIBE) schemes from CDH [15, 17]. The latter result has the same efficiency as the only (implicit) non-anonymous (H)IBE schemes from ϕ -hiding.

For constructing AIBE from CDH, we extend CE [17] to support blindness [8], a property that is useful for realizing anonymity. We then construct a blind CE scheme from CDH and adapt the non-blind IBE-from-CE framework [17] to obtain AIBE. For AHIBE from CDH and A(H)IBE from ϕ -hiding, we extend HE [18] and OTSE [15] to support blindness. We then propose a blind HE scheme from ϕ -hiding. With our blind HE and CE constructions, we further adapt the non-blind OTSE-from-HE/CE [18] and (H)IBE-from-OTSE [15] frameworks to obtain anonymous (H)IBE.

1.2 Technical Overview

Typically, IBE is realized by assigning each possible identity a unique public key, which can be done without knowing the corresponding master secret key. The small master public key essentially compresses all these public keys.

Non-black-box Approach to IBE. CE and OTSE are (possibly symmetric-key) encryption primitives associated with a hash function or a signature scheme. The hash or the signature serves as a compression of many possible inputs, ultimately, the public keys of underlying public-key encryption (PKE) to be compressed in the non-black-box IBE scheme [15, 17]. CE encryption takes in a hash value, an index, and a bit value. Decryption requires the witness (including the chameleon hash randomness) of a pre-image corresponding to the given hash value, where the bit at the specified index of the pre-image should match the bit specified by the encryptor. OTSE encryption shares the same interface, while decryption uses the signature as the witness, signing a message with its bit value at the specified index matching the bit specified by the encryptor.

A Merkle hash tree is built, with each node hashes or signs on two children nodes. In more detail, in the CE-based approach, each internal node is associated with a hash value of the values for its two children nodes, where the pre-image and the chameleon hash randomness form its secret key. For OTSE, the master public key contains the single verification key at the root, and the secret key for each node is a signature signing on the verification key of its two children nodes. For the leaf level, each node is associated with a PKE public/private key pair. All the key material for each node (e.g., the chameleon hash randomness and the OTSE key pair) is derived from a pseudorandom function taking the same master secret seed and the identity associated with each node as the input.

To encrypt to an identity id , a sequence of circuits will be garbled along the path from the leaf node for id to the root, where the leaf circuit proceeds a PKE encryption of the plaintext, and each non-leaf circuit proceeds a hash encryption of input labels of its child circuit. Interestingly, these encryption procedures are deferred to the decryptor. At last, the ciphertext consists of the sequence of garbled circuits and the input labels of the root circuit. The decryptor with identity id is given the PKE secret key for id and the corresponding decryption keys of CE/OTSE in its root-to-leaf path, which suffice to evaluate each garbled circuit and decrypt the output along the root-to-leaf path and eventually the plaintext.

Blinding Technique. The seminal non-black-box IBE scheme of Döttling and Garg [17] is not anonymous since invalid decryption reveals a prefix of the targeted recipient id – Given a ciphertext for id , a secret key for id' can evaluate the garbled circuits successfully up to the node of divergence between id and id' .

Brakerski *et al.* [8] suggested a blinding technique to force a (no matter whether valid or invalid) decryptor to go down the hash tree blindly. Namely, every decryptor is able to evaluate all the garbled circuits until the very end. Whereas an anonymous encryption scheme only requires a ciphertext to hide its recipient from unauthorized decryptors, Brakerski *et al.* proposed a stronger

blindness notion for IBE (and the underlying encryption and garbling primitives), which requires that a ciphertext/garbled circuit of a random input looks (partially) random to even authorized decryptors/evaluators. With such blind primitives, invalid evaluations in a non-black-box scheme always remain executable up to the end; otherwise, the decryptor breaks the blindness of the primitives. Moreover, one may observe that the blindness property is transmittable – the randomness property preserves for a blind ciphertext/garbled circuit of another blind ciphertext/garbled circuit. Thus, the blindness property of the underlying primitives directly leads to that of the non-black-box IBE scheme.

Limitation of the Blinding Technique. The transmittable blindness property seems customized for the non-black-box approach, which employs underlying primitives in a recursive manner. However, a blindness notion requires a strong definition, which is hard to achieve for most encryption primitives [15, 17, 18, 24] of non-black-box IBE. Namely, an encryption scheme is blind if each ciphertext is composed of a recipient-dependent part that looks random as long as the plaintext is random, and a recipient-independent part that only relates to the public information. Take the CE primitive [17], which allows encryption upon an index, as an example. Currently, there are two ways to construct CE: the accumulation-style framework [24] and the missing-block framework [17]. In both frameworks, the ciphertext must contain the input index plainly, which can be neither random nor only related to public information. It remains unclear how to anonymize other non-black-box IBE schemes using the blinding technique.

Anonymizing Döttling–Garg IBE. Our idea is to hide the input index within our blind IBE construction by including dummy components, similar to private broadcast encryption. We propose a blind CE scheme with relaxed blindness, in which a CE ciphertext consists of a random recipient-dependent part and an only-index-dependent part. The latter hides all non-public inputs of an encryption except the index, namely, the hash value, the expected bit at the index, and the plaintext.

In our blind IBE scheme, the first part of our CE ciphertexts will be generated in blind garbled circuits, which are “blindness-preserving” [8]. We can afford to put the second part of the CE ciphertexts directly in an IBE ciphertext (without deferred encryption via garbled circuits). A legitimate decryptor who has the secret key for the right identity will pick the correct ciphertext at the correct index. Meanwhile, using the wrong one, the CE decryption result will be random, which the blind garbled circuit will allow the evaluation continues until the leaf node, eventually decrypts to a random plaintext.

Anonymizing HIBE with Relaxed Blind CE. Recall that the IBE-from-CE construction [17] associates a hash key to each level of the tree. Extending it to support key delegation, a trivial approach would release the corresponding trapdoor, compromising the security of the whole level. As a remedy, a node-specific hash key, which is derived from the PRF as usual, would be introduced solely for

this purpose. (The usual “certification” of the two lower-level hash keys for the 0- and 1-branches would depend on the hash value from the above hash key.)

Unfortunately, our relaxed blind CE idea does not work with the HIBE-from-CE construction [17]. The reason is that the corresponding HIBE encryption now needs the node-specific hash key to encrypt, which will be done via deferred encryption, and is unavailable from the short master public key for encryptions to create the only-index-dependent part (outside of the garbled circuit).

We thus resort to the alternative non-black-box HIBE construction [18] from one-time signature with encryption (OTSE) [15]. OTSE certifies the two lower-level OTSE verification keys via one-time signatures, instead of (chameleon) hashing in CE. Its encryption interface remains the same as CE, except that the hash key is replaced with the OTSE verification key. To decrypt, it requires the one-time signature as the witness.

Instantiating this design takes three steps [18]. Non-compact OTSE, with the verification key size bigger than the size of the messages to be signed, can be built easily from PKE. A compact OTSE, akin to how the master public key is compact when (identity-based secret key extraction of) IBE is viewed as a signature scheme, can be built from a non-compact one via the help of hash encryption (HE). HE simply removes the collision tractability from the chameleon hash encryption. Indeed, this part could be seen as a generalization of the original approach [17]. (H)IBE scheme can then be built from a compact OTSE.

We observe that this design is compatible with our blind CE approach. We thus obtain an AHIBE scheme from CDH using our CDH-based blind CE. Finally, with one-way function with encryption (OWFE) [24], which essentially replaces CE/HE/OTSE with the witness being the pre-image of the associated OWF, we obtain AIBE and AHIBE constructions from the ϕ -hiding assumption, thanks to the existing OWFE construction from ϕ -hiding [24].

Organization. Section 2 defines notations, hard problems, hardcore predicates, A(H)IBE, CE, blind PKE, blind garbled circuits, and delegatable pseudorandom functions. Section 3 presents our blind CE construction. Sections 4 and 5 present our construction of A(H)IBE from CDH. Section 6 shows how to construct A(H)IBE from the ϕ -hiding assumption. Section 7 discusses DDH/LWE-version of our constructions. Our security proofs are in Appendix A.

2 Background

Let λ be the security parameter. PPT refers to probabilistic polynomial time. The notation $[n]$ denotes the set $\{1, \dots, n\}$, $\text{str}[i]$ denotes the i -th bit of str , and $\text{str}[\leq i]$ denotes the first i bits of str , particularly, $\text{str}[\leq 0] = \epsilon$ is the empty string.

2.1 Hard Problems and Hardcore Predicates

Definition 1 (Hard-to-Compute Functions). Let $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a function and $X = \{X_\lambda\}_\lambda$ be an ensemble of input distributions for f . f is

hard-to-compute for X if it holds for any PPT algorithm \mathcal{A} that $\Pr[\mathcal{A}(1^\lambda, x) = f(x)] \leq \text{negl}(\lambda)$, where $x \leftarrow X_\lambda$.

Definition 2 (Computational Diffie–Hellman (CDH) Assumption). Let (\mathbf{G}, \cdot) be a cyclic group of order p with generator g and $\text{CDH}(g, g^a, g^b) = g^{ab}$ for $a, b \xleftarrow{\$} \mathbf{Z}_p$. The CDH assumption states that CDH over \mathbf{G} is hard to compute.

A hard-to-compute function like the CDH problem by itself does not give any pseudorandom bit. Hardcore predicates associated with hard-to-compute functions upgrade a computational assumption into a decisional assumption. One can construct hardcore predicates for any hard-to-compute function [23].

Definition 3 (Hardcore Predicate [17]). Let $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a hard-to-compute function and $X = \{X_\lambda\}_\lambda$ be an ensemble of input distributions for f . A predicate $h : \{0, 1\}^* \rightarrow \{0, 1\}$ is a hardcore predicate for f if h is deterministic and efficiently computable given $f(x)$, and for any PPT algorithm \mathcal{A} : $\Pr[\mathcal{A}(1^\lambda, x) = h(f(x))] \leq 1/2 + \text{negl}(\lambda)$, where $x \leftarrow X_\lambda$.

Theorem 1 (Goldreich–Levin Theorem [23]). For every hard-to-compute function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$, there is a Goldreich–Levin hardcore predicate $g : \{0, 1\}^* \rightarrow \{0, 1\}$ for f , which is only a gentle modification of f , such that the hardness of g is equivalent to that of f .

2.2 Anonymous (Hierarchical) Identity-Based Encryption

Definition 4 (Identity-Based Encryption). An IBE scheme consists of four PPT algorithms (Setup, KeyGen, Enc, Dec) as follows:

- $\text{Setup}(1^\lambda, 1^n)$: This algorithm takes as input a security parameter 1^λ and an identity length 1^n , and it outputs a master public key mpk and a master secret key msk .
- $\text{KeyGen}(\text{mpk}, \text{msk}, \text{id})$: This algorithm takes as input the master public key mpk , the master secret key msk , and an identity id , and it outputs a secret key sk_{id} .
- $\text{Enc}(\text{mpk}, \text{id}, m; r)$: This algorithm takes as input the master public key mpk , an identity id , a message m , and a randomness r , and it outputs a ciphertext ct .
- $\text{Dec}(\text{mpk}, \text{sk}_{\text{id}}, \text{ct})$: This algorithm takes as input the master public key mpk , a secret key sk_{id} , a ciphertext ct , and it outputs a plaintext m or \perp .

We require that an IBE scheme satisfies the following two properties:

- *Correctness:* $\text{Dec}(\text{mpk}, \text{sk}_{\text{id}}, \text{ct}) = m$ with probability 1, where $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda, 1^n)$, $\text{sk}_{\text{id}} \leftarrow \text{KeyGen}(\text{mpk}, \text{msk}, \text{id})$, and $\text{ct} \leftarrow \text{Enc}(\text{mpk}, \text{id}, m)$, over the randomness of (Setup, KeyGen, Enc, Dec).
- *IND-ID-CPA Security:* Any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:
 1. $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda, 1^n)$

2. $(id^*, m_0, m_1, st) \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot)}(\text{mpk})$
3. $\zeta \xleftarrow{\$} \{0, 1\}, ct \leftarrow \text{Enc}(\text{mpk}, id^*, m_\zeta)$
4. $\zeta' \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot)}(st, ct)$
5. \mathcal{A} wins if $\zeta' = \zeta$ and \mathcal{A} never queried id^* to its KeyGen oracle.

Definition 5 (Anonymous Identity-Based Encryption). An AIBE scheme is an IBE scheme with the following stronger notion of security:

◦ *IND-ANON-ID-CPA Security:* Any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:

1. $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda, 1^n)$
2. $(id_0, id_1, m, st) \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot)}(\text{mpk})$
3. $\zeta \xleftarrow{\$} \{0, 1\}, ct \leftarrow \text{Enc}(\text{mpk}, id_\zeta, m)$
4. $\zeta' \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot)}(st, ct)$
5. \mathcal{A} wins if $\zeta' = \zeta$ and \mathcal{A} never queried id_0 or id_1 to its KeyGen oracle.

In HIBE, the identities are no longer of fixed length. The input 1^n to Setup defines the maximum length of identities. HIBE features an additional algorithm Delegate, which allows the generation of delegate keys dk. The delegate key dk_{id} for an identity id enables a user to generate secret (or delegate) keys using the KeyGen (or Delegate) algorithm for any identity with prefix id . The delegate key dk_ϵ for the empty identity ϵ is defined to be the master secret key msk .

Definition 6 ((Selective-ID) Hierarchical Identity-Based Encryption).

An HIBE scheme is an IBE scheme with an additional Delegate algorithm:

- $\text{Delegate}(\text{mpk}, dk_{id}, id')$: This algorithm takes as input master public key mpk , a delegate key dk_{id} and an identity id' , and it outputs a delegate key $dk_{id||id'}$.

We require that an HIBE scheme satisfies the following two properties:

- *Correctness:* We have $\text{Delegate}(\text{mpk}, \text{msk}, id||id') = \text{Delegate}(\text{mpk}, dk_{id}, id')$, $\text{KeyGen}(\text{mpk}, \text{msk}, id||id') = \text{KeyGen}(\text{mpk}, dk_{id}, id')$, and $\text{Dec}(\text{mpk}, sk_{id}, ct) = m$ hold with probability 1, $\forall \lambda \in \mathbb{N}^+$, where $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$, $sk_{id} \leftarrow \text{KeyGen}(\text{mpk}, \text{msk}, id)$, $dk_{id} \leftarrow \text{Delegate}(\text{mpk}, \text{msk}, id)$, and $ct \leftarrow \text{Enc}(\text{mpk}, id, m)$.
- *sel-IND-ID-CPA Security:* Any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:
 1. $id^* \leftarrow \mathcal{A}(1^\lambda)$
 2. $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$
 3. $(m_0, m_1, st) \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot), \text{Delegate}(\text{mpk}, \text{msk}, \cdot)}(\text{mpk})$
 4. $\zeta \xleftarrow{\$} \{0, 1\}, ct \leftarrow \text{Enc}(\text{mpk}, id^*, m_\zeta)$
 5. $\zeta' \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot), \text{Delegate}(\text{mpk}, \text{msk}, \cdot)}(st, ct)$
 6. \mathcal{A} wins if $\zeta' = \zeta$, \mathcal{A} never queried id^* to its KeyGen oracle, and never queried id to its Delegate oracle for some id which is a prefix of id^* .

Definition 7 ((Selective-ID) Anonymous Hierarchical Identity-Based Encryption). An AHIBE scheme is an HIBE scheme with the following stronger notion of security:

◦ *sel-IND-ANON-ID-CPA Security*: Any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:

1. $(\text{id}_0, \text{id}_1) \leftarrow \mathcal{A}(1^\lambda)$, it is required that $|\text{id}_0| = |\text{id}_1|$.
2. $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$
3. $(m, \text{st}) \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot), \text{Delegate}(\text{mpk}, \text{msk}, \cdot)}(\text{mpk})$
4. $\zeta \xleftarrow{\$} \{0, 1\}$, $\text{ct} \leftarrow \text{Enc}(\text{mpk}, \text{id}_\zeta, m)$
5. $\zeta' \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot), \text{Delegate}(\text{mpk}, \text{msk}, \cdot)}(\text{st}, \text{ct})$
6. \mathcal{A} wins if and only if $\zeta' = \zeta$ and \mathcal{A} never queried id_0 or id_1 to its KeyGen oracle, and \mathcal{A} never queried id to its Delegate oracle for some id which is a prefix of id_0 or id_1 .

2.3 Chameleon Encryption (CE)

A CE scheme [17] is a chameleon hash function that supports encryption and decryption procedures. It encrypts w.r.t. a hash value h and an index-bit pair (i, b) . The ciphertext can be decrypted given a preimage (x, r) of h where $x_i = b$.

Definition 8 (Chameleon Encryption [17]). A CE scheme consists of five PPT algorithms $(\text{KeyGen}, \text{H}, \text{H}^{-1}, \text{Enc}, \text{Dec})$ as follows:

- $\text{KeyGen}(1^\lambda, n)$: This algorithm takes as input the security parameter 1^λ and a message length n , and it outputs a key k and a trapdoor t .
- $\text{H}(k, x; r)$: This algorithm takes as input a key k , a message $x \in \{0, 1\}^n$, and coins r , and it outputs a hash value h , where h is of λ bits.
- $\text{H}^{-1}(t, (x, r), x')$: This algorithm takes as input a trapdoor t , a message $x \in \{0, 1\}^n$, coins r , and a message $x' \in \{0, 1\}^n$, and it returns r' .
- $\text{Enc}(k, (h, i, b), m; \rho)$: This algorithm takes as input a key k , a hash value h , an index $i \in [n]$, $b \in \{0, 1\}$, a message $m \in \{0, 1\}^*$, and a randomness ρ , and it outputs a ciphertext ct .
- $\text{Dec}(k, (x, r), \text{ct})$: This algorithm takes as input a key k , a message x , coins r , and a ciphertext ct , and it outputs a value m (or \perp).

We require that a CE scheme satisfies the following properties:

- *Uniformity*: $\forall x, x' \in \{0, 1\}^n$, the distributions $\text{H}(k, x; r)$ and $\text{H}(k, x'; r')$ are statistically close, where r, r' are chosen uniformly at random.
- *Trapdoor Collisions*: For any $x, x' \in \{0, 1\}^n$, and r , if $(k, t) \leftarrow \text{KeyGen}(1^\lambda, n)$ and $r' = \text{H}^{-1}(t, (x, r), x')$, then $\text{H}(k, x; r) = \text{H}(k, x'; r')$. Moreover, if r is chosen uniformly at random, then r' is also statistically close to uniform.
- *Correctness*: $\text{Dec}(k, (x, r), \text{ct}) = m$ holds $\forall \lambda$ with probability 1 where $(k, t) \leftarrow \text{KeyGen}(1^\lambda, n)$, $h = \text{H}(k, x; r)$, and $\text{ct} \leftarrow \text{Enc}(k, (h, i, x_i), m)$, over the randomness of $(\text{KeyGen}, \text{H}, \text{H}^{-1}, \text{Enc}, \text{Dec})$.
- *IND Security*: Any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:
 1. $(k, t) \leftarrow \text{KeyGen}(1^\lambda, n)$
 2. $(x, r, i, \text{st}) \leftarrow \mathcal{A}(k)$
 3. $\zeta \xleftarrow{\$} \{0, 1\}$, $\text{ct} \leftarrow \text{Enc}(k, (\text{H}(k, x; r), i, 1 - x_i), \zeta)$
 4. $\zeta' \leftarrow \mathcal{A}(\text{st}, \text{ct})$. \mathcal{A} wins if $\zeta' = \zeta$.

2.4 Blind Public Key Encryption

In blind PKE [8], an encryption of a random plaintext is (partially) indistinguishable from random, even when the corresponding secret key is known.

Definition 9 (Blind Public Key Encryption (with public parameters)). A blind PKE scheme consists of four PPT algorithms (Init, KeyGen, Enc, Dec):

- $\text{Init}(1^\lambda)$: This algorithm takes as input a security parameter 1^λ , and it outputs public parameters pp .
- $\text{KeyGen}(\text{pp}; r)$: This algorithm takes as input the public parameters pp , and a randomness r , and it outputs a pair of public and secret keys (pk, sk) .
- $\text{Enc}(\text{pp}, \text{pk}, m; \rho)$: This algorithm takes as input the public parameters pp , a public key pk , a message m , and a randomness ρ , and it outputs a ciphertext ct .
- $\text{Dec}(\text{pp}, \text{sk}, \text{ct})$: This algorithm takes as input the public parameters pp , a secret key sk , and a ciphertext ct , and it outputs a plaintext m or \perp .

We require that a blind PKE scheme satisfies the following three properties:

- o *Correctness*: $\text{Dec}(\text{pp}, \text{sk}, \text{ct}) = m$ holds $\forall \lambda$ with probability 1, where $\text{pp} \leftarrow \text{Init}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$, and $\text{ct} \leftarrow \text{Enc}(\text{pp}, \text{pk}, m)$, over the randomness of (Init, KeyGen, Enc, Dec).
- o *IND-CPA Security*: Any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:
 1. $\text{pp} \leftarrow \text{Init}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$
 2. $(m_0, m_1, \text{st}) \leftarrow \mathcal{A}(\text{pp}, \text{pk})$
 3. $\zeta \xleftarrow{\$} \{0, 1\}$, $\text{ct} \leftarrow \text{Enc}(\text{pp}, \text{pk}, m_\zeta)$
 4. $\zeta' \leftarrow \mathcal{A}(\text{st}, \text{ct})$. \mathcal{A} wins if $\zeta' = \zeta$.
- o *IND-BLIND-CPA Security*: We can decompose the output of $\text{Enc}(\text{pp}, \text{pk}, m; \rho)$ into $\text{E}_1(\text{pp}; \rho) \parallel \text{E}_2(\text{pp}, \text{pk}, m; \rho)$ such that any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:
 1. $\text{pp} \leftarrow \text{Init}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$
 2. $m \xleftarrow{\$} \mathcal{M}$, $\rho \xleftarrow{\$} \mathcal{R}$ (\mathcal{M} and \mathcal{R} are the message and randomness spaces)
 3. $\bar{\text{ct}}_1 = \text{E}_1(\text{pp}; \rho)$, $\bar{\text{ct}}_2 = \text{E}_2(\text{pp}, \text{pk}, m; \rho)$
 4. $L = |\bar{\text{ct}}_2|$, $\bar{\text{ct}}'_2 \xleftarrow{\$} \{0, 1\}^L$, $\zeta \xleftarrow{\$} \{0, 1\}$
 5. if $\zeta = 0$, then $\text{ct} = (\bar{\text{ct}}_1, \bar{\text{ct}}_2)$; if $\zeta = 1$, then $\text{ct} = (\bar{\text{ct}}_1, \bar{\text{ct}}'_2)$
 6. $\zeta' \leftarrow \mathcal{A}(\text{pp}, \text{pk}, \text{sk}, \text{ct})$. \mathcal{A} wins if $\zeta' = \zeta$.

2.5 Blind Garbled Circuits

A blind garbling scheme [8] is a garbling scheme with a blindness security property, which requires that a simulator’s output is uniformly random as long as the corresponding input is uniformly random.

Definition 10 (Blind Garbling Scheme [8]). A blind garbling scheme consists of three PPT algorithms (Garble, Eval, Sim) as follows:

- $\text{Garble}(1^\lambda, C)$: This algorithm takes as input a security parameter 1^λ and a circuit $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$, and it outputs a garbled circuit \hat{C} along with input labels $(\text{lbl}_{i,b})_{i \in [n], \zeta \in \{0,1\}}$ where each label $\text{lbl}_{i,b} \in \{0, 1\}^\lambda$.
- $\text{Eval}(\hat{C}, \hat{L})$: This algorithm takes as input a garbled circuit \hat{C} along with a set of n labels $\hat{L} = (\text{lbl}_{i,x_i})_{i \in [n]}$ for some string $x \in \{0, 1\}^n$, and it outputs a string $y \in \{0, 1\}^m$.
- $\text{Sim}(1^{|\mathcal{C}|}, 1^n, y)$: This algorithm takes as input the description length of C , input length n , and an m -bit string y , and it outputs a simulated garbled circuit \tilde{C} and labels $\tilde{L} = (\text{lbl}_i)_{i \in [n]}$.

We require that a blind garbling scheme satisfies the following three properties:

- o *Correctness*: For all circuits $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$, inputs $x \in \{0, 1\}^n$, $(\hat{C}, (\text{lbl}_{i,b})_{i,b}) \leftarrow \text{Garble}(1^\lambda, C)$, and $\hat{L} = (\text{lab}_{i,x_i})_{i \in [n]}$, we have $\text{Eval}(\hat{C}, \hat{L}) = C(x)$.
- o *Simulation Security*: For all circuits $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$, inputs $x \in \{0, 1\}^n$, we have that the distribution $\{(\hat{C}, \hat{L}) : (\hat{C}, (\text{lbl}_{i,b})_{i,b}) \leftarrow \text{Garble}(1^\lambda, C), \hat{L} = (\text{lbl}_{i,x_i})_{i \in [n]}\}$ is computationally indistinguishable from the distribution $\{(\tilde{C}, \tilde{L}) : (\tilde{C}, \tilde{L}) \leftarrow \text{Sim}(1^{|\mathcal{C}|}, 1^n, C(x))\}$.
- o *IND-BLIND Security*: For all circuits $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$ and a uniformly random output $y \in \{0, 1\}^m$, the distribution $\{(\tilde{C}, \tilde{L}) \leftarrow \text{Sim}(1^{|\mathcal{C}|}, 1^n, y)\}$ is indistinguishable from a completely uniform bit string of the same length.

2.6 Delegatable Pseudorandom Functions

Pseudorandom functions (PRF) are efficiently computable seeded functions that are indistinguishable from truly random functions under oracle access. Delegatable PRF [17] is PRF that additionally supports the delegation of seeds for inputs that start with certain prefixes. The classic GGM construction [22] can be made delegatable. We use PRF in our AIBE, and delegatable PRF for AHIBE.

Definition 11 (Pseudorandom Function). $\mathcal{F} : \{0, 1\}^\lambda \times \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ is a PRF if it holds for any PPT algorithm \mathcal{A} that

$$|\Pr[\mathcal{A}^{\mathcal{F}(s,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1]| < \text{negl}(\lambda),$$

where \mathcal{F} is efficiently computable, $s \in \{0, 1\}^\lambda$ is a seed for \mathcal{F} , and $f : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ is a uniformly random function.

Definition 12 (Delegatable Pseudorandom Function). A delegatable PRF consists of two PPT algorithms $(\mathcal{F}, \mathcal{F}.\text{Delegate})$ as follows:

- $\mathcal{F}(s, x)$ This algorithm takes as input a seed $s \in \{0, 1\}^\lambda$ and a string $x \in \{0, 1\}^*$, and it outputs a value $u \in \{0, 1\}^\lambda$.
- $\mathcal{F}.\text{Delegate}(s, x)$ This algorithm takes as input a seed s and an input x , and it outputs a seed s_x .

We require that a delegatable PRF satisfies the following two properties:

- *Delegatability:* For all inputs $x, x' \in \{0, 1\}^*$, we have $\mathcal{F}(s, x||x') = \mathcal{F}(s_x, x')$, where $s_x = \mathcal{F}.\text{Delegate}(s, x)$.
- *Selective Pseudorandomness:* For any PPT algorithm \mathcal{A} and every $x \in \{0, 1\}^*$ of size at most polynomial in λ , we have that

$$|\Pr[\mathcal{A}^{\mathcal{F}(s, \cdot), \text{Delegate}(s, \cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{H(\cdot), \text{Delegate}(s, \cdot)}(1^\lambda) = 1]| \leq \text{negl}(\lambda),$$

where $s \leftarrow \{0, 1\}^\lambda$ is a seed for \mathcal{F} , $\text{Delegate}(s, \cdot)$ delegates seeds for all inputs $x' \in \{0, 1\}^*$ that are not x 's prefix, and H is a function that is uniformly random on x and all its prefixes, but identical to $\mathcal{F}(s, \cdot)$ on all other inputs.

3 Blind Chameleon Encryption

We formulate blind CE. A blind CE encryption of a random plaintext is indistinguishable from random even to an adversary who has the decryption power.

3.1 Definition

In this paper, a relaxed notion of blindness suffices, which only expects a part of the ciphertext to be indistinguishable from random.

Definition 13 (Blind Chameleon Encryption). *A CE scheme is blind if it satisfies the following stronger notion of security:*

- *IND-BLIND Security:* There exists some decomposition of $\text{Enc}(k, (h, i, b), m; \rho) = \text{E}_1(k, i; \rho) || \text{E}_2(k, (h, i, b), m; \rho)$ such that any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:
 1. $(k, t) \leftarrow \text{KeyGen}(1^\lambda, l)$
 2. $(x, r, i, \text{st}) \leftarrow \mathcal{A}(k)$
 3. $m \leftarrow \{0, 1\}^l, \rho \xleftarrow{\$} \mathcal{R}$
 4. $\bar{\text{ct}}_1 = \text{E}_1(k, i; \rho), \bar{\text{ct}}_2 = \text{E}_2(k, (H(k, x; r), i, x_i), m; \rho)$
 5. $L = |\bar{\text{ct}}_2|, \bar{\text{ct}}'_2 \xleftarrow{\$} \{0, 1\}^L, \zeta \xleftarrow{\$} \{0, 1\}$
 6. if $\zeta = 0$, then $\text{ct} = (\bar{\text{ct}}_1, \bar{\text{ct}}_2)$; if $\zeta = 1$, then $\text{ct} = (\bar{\text{ct}}_1, \bar{\text{ct}}'_2)$
 7. $\zeta' \leftarrow \mathcal{A}(\text{st}, \text{ct})$. \mathcal{A} wins if $\zeta' = \zeta$.

3.2 Our Construction

We modify the CE scheme of Döttling and Garg [17] to obtain a blind CE scheme, with modifications highlighted in red. In particular, while the component e in the ciphertext is random as long as m is random, we let the other part of the ciphertext be independent of the targeted hash value h , by pushing h^ρ into e . Intuitively, the key k contains $2n$ random group elements $g_{j,b'}$, each corresponding to one unique index-bit pair. A hash value h of a string x is the multiplication of group elements g_{j,x_j} corresponding to each bit j of x . To encrypt to a hash value h and an index-bit (i, b) pair, the value h divided by $g_{i,b}$ is used to hide the plaintext, while group elements $g_{j,b'}$ corresponding to all

other indexes $j \neq i$ are given. With the preimage x of h s.t. $x_i = b$, a decryptor is able to compute h divided by $g_{i,b}$ using the group elements corresponding to other indexes, and finally recover the plaintext.

Let (\mathbf{G}, \cdot) be a cyclic group of order p with generator g . Let f be a hard-to-compute function defined as $f(g, g^a, g^b, g^c) = g^{(a-b) \cdot c}$. It is easy to see that the hardness of f is equivalent to that of CDH over \mathbf{G} . Let $\text{HardCore}(\cdot)$ be the Goldreich–Levin hardcore predicate of f . Our blind CE construction is as follows.

- $\text{KeyGen}(1^\lambda, n)$:
 - For each $j \in [n]$, $b' \in \{0, 1\}$, choose a uniformly random value $\alpha_{j,b'} \leftarrow \mathbf{Z}_p$ and compute $g_{j,b'} = g^{\alpha_{j,b'}}$.
 - Output $k = (g, (g_{j,b'})_{j \in [n], b' \in \{0,1\}})$ and $t = ((\alpha_{j,b'})_{j \in [n], b' \in \{0,1\}})$.
- $\text{H}(k, x; r)$: Sample $r \leftarrow \mathbf{Z}_p$ and output $h = g^r \cdot \prod_{j \in [n]} g_{j,x_j}$.
- $\text{H}^{-1}(t, (x, r), x')$: Output $r' = r + \sum_{j \in [n]} (\alpha_{j,x_j} - \alpha_{j,x'_j}) \bmod p$.
- $\text{Enc}(k, (h, i, b), m; \rho)$:
 - Sample $\rho \leftarrow \mathbf{Z}_p$.
 - For $j \neq i$, $b' \in \{0, 1\}$, compute $c_{j,b'} = g_{j,b'}^\rho$.
 - Compute $c = g^\rho$ and $e = \text{HardCore}(\frac{h^\rho}{g_{p,b}}) \oplus m$.
 - Output $\text{ct} = (\bar{\text{ct}}_1 = (i, c, (c_{j,b'})_{j,b'}), \bar{\text{ct}}_2 = e)$.
- $\text{Dec}(k, (x, r), \text{ct})$: Output $m = \text{HardCore}(c^r \cdot \prod_{j \in [n] \setminus \{i\}} c_{j,x_j}) \oplus e$.

Efficiency. The efficiency of our CE construction is almost the same as that of Döttling and Garg [17], except that we move one multiplication operation from the encryption procedure to the decryption procedure, whereas each encryption procedure performs $O(n)$ exponentiations, and each decryption procedure performs $O(n)$ multiplications, where n is the input length to KeyGen .

Theorem 2 ([17]). *The construction described above is chameleon encryption if the CDH problem over \mathbf{G} is hard.*

Theorem 3. *The given chameleon encryption scheme is blind, where E_1 outputs $\bar{\text{ct}}_1$ and E_2 outputs $\bar{\text{ct}}_2$.*

The proof is deferred to Appendix A.1.

4 Anonymous Identity-Based Encryption

We give our construction of AIBE scheme from CDH, as a consequence of upgrading the Döttling–Garg IBE scheme [17] to blind IBE.

4.1 Blind Identity-Based Encryption

A blind IBE encryption of a random plaintext is (partially) indistinguishable from random, even when the corresponding secret key is known.

Definition 14 (Blind Identity-Based Encryption [8]). *An IBE scheme is blind if it satisfies the following stronger notion of security:*

- *IND-BLIND-ID-CPA Security:* *There exists some decomposition of $\text{Enc}(\text{mpk}, \text{id}, m; \rho) = \text{E}_1(\text{mpk}; \rho) \parallel \text{E}_2(\text{mpk}, \text{id}, m; \rho)$ such that any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:*
 1. $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda, 1^n)$
 2. $(\text{id}^*, \text{st}) \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot)}(\text{mpk})$
 3. $m \xleftarrow{\$} \mathcal{M}, \rho \xleftarrow{\$} \mathcal{R}$ (\mathcal{M} and \mathcal{R} are the plaintext and randomness spaces)
 4. $\text{ct}_1 = \text{E}_1(\text{mpk}; \rho), \text{ct}_2 = \text{E}_2(\text{mpk}, \text{id}^*, m; \rho)$
 5. $L = |\text{ct}_2|, \text{ct}'_2 \xleftarrow{\$} \{0, 1\}^L, \zeta \xleftarrow{\$} \{0, 1\}$
 6. *if $\zeta = 0$, then $\text{ct} = (\text{ct}_1, \text{ct}_2)$; if $\zeta = 1$, then $\text{ct} = (\text{ct}_1, \text{ct}'_2)$*
 7. $\zeta' \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot)}(\text{st}, \text{ct})$. *\mathcal{A} wins if $\zeta' = \zeta$.*

Theorem 4 ([8]). *A blind IBE scheme is also an AIBE scheme.*

4.2 Our Construction

We construct a blind IBE scheme from our blind CE scheme, basing on Döttling–Garg IBE [17], with modifications highlighted in red.

Let \mathcal{CE} be a blind CE scheme, $\mathcal{PK}\mathcal{E}$ be a blind PKE scheme, where the hash value of \mathcal{CE} and the public keys of $\mathcal{PK}\mathcal{E}$ have the same length l polynomial in λ , \mathcal{GC} be a blind garbling scheme, and \mathcal{F} be a pseudorandom function.

We first define two functions **NodeGen** and **LeafGen**, and three circuits **T**, **Q**, and **BatchEnc**. The **NodeGen** and **LeafGen** functions define the Merkle hash tree (for non-leaf and leaf nodes, respectively) and provide access to the corresponding CE and PKE keys. The **T** (resp., **Q**) circuit acts as the leaf (resp., non-leaf) circuit proceeding PKE (resp., CE) encryption of the plaintext (resp., input to its child circuit). The **BatchEnc** circuit proceeds a set of recipient-dependent CE encryptions, which will be called in the **Q** circuit. The main purpose for us to define the additional function **BatchEnc** instead of directly defining **Q** is for easier replacement of **Q** in our security proofs.

- **NodeGen** $((k_0, \dots, k_{n-1}), (t_0, \dots, t_{n-1}, s), v)$:
 - Let $i = |v|$; compute $h_v = \mathcal{CE.H}(k_i, 0^{2l}; \mathcal{F}(s, v))$.
 - Compute $h_{v\parallel 0} = \mathcal{CE.H}(k_{i+1}, 0^{2l}; \mathcal{F}(s, v\parallel 0))$.
 - Compute $h_{v\parallel 1} = \mathcal{CE.H}(k_{i+1}, 0^{2l}; \mathcal{F}(s, v\parallel 1))$.
 - Let $x_v = h_{v\parallel 0} \parallel h_{v\parallel 1}$; compute $r_v = \mathcal{CE.H}^{-1}(t_i, (0^{2l}, \mathcal{F}(s, v)), x_v)$.
 - Output (h_v, x_v, r_v) .
- **LeafGen** $((\text{pp}, k_{n-1}), (t_{n-1}, s), v)$:
 - Compute $h_v = \mathcal{CE.H}(k_{n-1}, 0^{2l}; \mathcal{F}(s, v))$.
 - Compute $(\text{lpk}_{v\parallel 0}, \text{lsk}_{v\parallel 0}) = \mathcal{PK}\mathcal{E}.\text{KeyGen}(\text{pp}; \mathcal{F}(s, v\parallel 0))$.
 - Compute $(\text{lpk}_{v\parallel 1}, \text{lsk}_{v\parallel 1}) = \mathcal{PK}\mathcal{E}.\text{KeyGen}(\text{pp}; \mathcal{F}(s, v\parallel 1))$.
 - Let $x_v = \text{lpk}_{v\parallel 0} \parallel \text{lpk}_{v\parallel 1}$; compute $r_v = \mathcal{CE.H}^{-1}(t_{n-1}, (0^{2l}, \mathcal{F}(s, v)), x_v)$.
 - Output $((h_v, x_v, r_v), \text{lsk}_{v\parallel 0}, \text{lsk}_{v\parallel 1})$.

- $T[\text{pp}, m, \rho](\text{lpk})$: Compute and output $\mathcal{PK}\mathcal{E}.E_2(\text{pp}, \text{lpk}, m; \rho)$.
- $Q[k, \beta, \bar{Y}, \bar{\rho}](h)$: Compute and output $\text{BatchEnc}(k, h, \beta, \bar{Y}; \bar{\rho})$.
- $\text{BatchEnc}(k, h, \beta, \bar{Y}; \bar{\rho})$: This function takes input as a chameleon encryption key k , a hash value h , a flag $\beta \in \{0, 1\}$, a set of labels $\bar{Y} = (Y_{j,b})_{j \in [l], b \in \{0,1\}}$, and a set of randomness $\bar{\rho} = (\rho_{j,b})_{j \in [2l], b \in \{0,1\}}$.
 - For $j \in [l], b \in \{0, 1\}$, derive $\text{ct}_{j,b} = \mathcal{CE}.E_2(k, (h, j + \beta \cdot l, b), Y_{j,b}; \rho_{j+\beta \cdot l, b})$.
 - Output $(\text{ct}_{j,b})_{j \in [l], b \in \{0,1\}}$.

Below presents our blind IBE scheme (Setup, KeyGen, Enc, Dec).

- Setup($1^\lambda, 1^n$):
 - Compute $\text{pp}^{\mathcal{PK}\mathcal{E}} \leftarrow \mathcal{PK}\mathcal{E}.\text{Init}(1^\lambda)$.
 - For each $i \in [0, n-1]$, compute $(k_i, t_i) \leftarrow \mathcal{CE}.\text{KeyGen}(1^l, 2l)$.
 - Choose a random seed s for \mathcal{F} .
 - Let $v_0 = \epsilon$ be the root node; compute $(h_0, \cdot, \cdot) = \text{NodeGen}((k_0, \dots, k_{n-1}), (t_0, \dots, t_{n-1}, s), v_0)$.
 - Output $\text{mpk} = (\text{pp}^{\mathcal{PK}\mathcal{E}}, k_0, \dots, k_{n-1}, h_0)$ and $\text{msk} = (t_0, \dots, t_{n-1}, s)$.
- KeyGen($\text{mpk}, \text{msk}, \text{id}$):
 - Define $v_i = \text{id}[\leq i]$ for $i \in [0, n-1]$.
 - For $i \in [0, n-2]$, compute $(h_i, x_i, r_i) = \text{NodeGen}((k_0, \dots, k_{n-1}), (t_0, \dots, t_{n-1}, s), v_i)$.
 - For $i = n-1$, compute $((h_i, x_i, r_i), \text{lsk}_{v_i \| 0}, \text{lsk}_{v_i \| 1}) = \text{LeafGen}((\text{pp}^{\mathcal{PK}\mathcal{E}}, k_{n-1}), (t_{n-1}, s), v_i)$.
 - Output $\text{sk}_{\text{id}} = ((h_0, x_0, r_0), \dots, (h_{n-1}, x_{n-1}, r_{n-1}), \text{lsk}_{\text{id}})$.
- Enc($\text{mpk}, \text{id}, m; (\rho', \bar{\rho}^{(1)}, \dots, \bar{\rho}^{(n)})$):
 - Sample random strings $\rho', \bar{\rho}^{(1)}, \dots, \bar{\rho}^{(n)}$ where $\bar{\rho}^{(i)} = (\rho_{j,b}^{(i)})_{j \in [2l], b \in \{0,1\}}$.
 - Compute $(\tilde{T}, \bar{Y}^{(n)}) \leftarrow \mathcal{GC}.\text{Garble}(1^\lambda, T[\text{pp}^{\mathcal{PK}\mathcal{E}}, m, \rho'])$.
 - Compute $\text{ct}^{(n)} = \mathcal{PK}\mathcal{E}.E_1(\text{pp}^{\mathcal{PK}\mathcal{E}}; \rho')$.
 - For $i = n-1, \dots, 0$, compute $(\tilde{Q}^{(i)}, \bar{Y}^{(i)}) \leftarrow \mathcal{GC}.\text{Garble}(1^\lambda, Q[k_i, \text{id}[i+1], \bar{Y}^{(i+1)}, \bar{\rho}^{(i+1)}])$ and $\text{ct}^{(i)} = (\text{ct}_{j,b}^{(i)})_{j,b} = (\mathcal{CE}.E_1(k_i, j; \rho_{j,b}^{(i+1)}))_{j \in [2l], b \in \{0,1\}}$.
 - Parse $\bar{Y}^{(0)} = (Y_{j,b}^{(0)})_{j,b}$, let $\tilde{Y}^{(0)} = (Y_{j, h_0[j]}^{(0)})_j$.
 - Output $\text{ct} = (\text{ct}^{(0)}, \dots, \text{ct}^{(n)}, \tilde{Q}^{(0)}, \dots, \tilde{Q}^{(n-1)}, \tilde{T}, \tilde{Y}^{(0)})$.
- Dec($\text{mpk}, \text{sk}_{\text{id}}, \text{ct}$):
 - For $i = 0, \dots, n-1$, proceed as follows:
 - * Compute $\bar{\text{ct}}^{(i)'} = \mathcal{GC}.\text{Eval}(\tilde{Q}^{(i)}, \tilde{Y}^{(i)})$, parse $\bar{\text{ct}}^{(i)'} = (\text{ct}_{j,b}^{(i)'})_{j \in [l], b \in \{0,1\}}$.
 - * For $j \in [l]$, compute $\text{ct}_j^{(i)} = \text{ct}_{j', x_i[j']}^{(i)} \parallel \text{ct}_{j, x_i[j']}^{(i)'}$ and $\tilde{y}_j^{(i+1)} = \mathcal{CE}.\text{Dec}(k_i, (x_i, r_i), \text{ct}_j^{(i)})$ where $j' = j + \text{id}[i+1] \cdot l$.
 - * Let $\tilde{Y}^{(i+1)} = (\tilde{y}_j^{(i+1)})_{j \in [l]}$.
 - Compute $\text{ct}^{(n)'} = \mathcal{GC}.\text{Eval}(\tilde{T}, \tilde{Y}^{(n)})$.
 - Output $m = \mathcal{PK}\mathcal{E}.\text{Dec}(\text{pp}^{\mathcal{PK}\mathcal{E}}, \text{lsk}_{\text{id}}, \text{ct}^{(n)} \parallel \text{ct}^{(n)'})$.

Correctness. Consider a secret key $\text{sk}_{\text{id}} = ((h_0, x_0, r_0), \dots, (h_{n-1}, x_{n-1}, r_{n-1}))$, lsk_{id} , and a ciphertext corresponding to the same identity $\text{ct} = (\text{ct}^{(0)}, \dots, \text{ct}^{(n)})$, $\tilde{Q}^{(0)}, \dots, \tilde{Q}^{(n-1)}, \tilde{T}, \tilde{Y}^{(0)}$. We consider each step of the decryption process below.

By correctness of \mathcal{GC} , the evaluation of $\tilde{Q}^{(0)}$ yields the output of BatchEnc that takes input as labels of the next garbled circuit $\tilde{Q}^{(1)}$. Next, by construction, we have the correct CE encryptions $(\text{ct}_j^{(1)})_j$ of the labels for the next garbled circuit $\tilde{Q}^{(1)}$. By correctness of \mathcal{CE} , the CE decryption of the appropriate ciphertexts yields the correct labels $(Y_{j, h_1[j]}^{(1)})_j$ for $\tilde{Q}^{(1)}$. Following the same argument, we can argue that the CE decryption of the appropriate ciphertexts generated by $\tilde{Q}^{(1)}$ yields the correct input labels for $\tilde{Q}^{(2)}$. Repeating this argument, the last garbled circuit \tilde{T} outputs a PKE encryption of m under lpk_{id} . Finally, the correctness of \mathcal{PKC} ensures that the recovered m is the encrypted one.

Efficiency. The most computationally intensive part of a non-black-box IBE scheme is the non-black-box use of underlying primitives inside garbled circuits during each decryption procedure. In this paper, we will only compare the efficiency among the non-black-box constructions.

Our AIBE construction shares almost the same efficiency as Döttling–Garg IBE, except that our encryption procedure additionally performs $O(2nl^2)$ exponentiations to run nl recipient-independent CE encryptions, and our ciphertext size is increased by $O(2nl^2)$ group elements to contain nl recipient-independent CE ciphertexts. These increases are minor compared to the overall encryption time and ciphertext size, which mainly depend on the time to garble circuits and the size of the garbled circuits, respectively.

Notably, our AIBE construction greatly improves the efficiency of the AIBE scheme of Brakerski *et al.* [8], as shown in Table 1, where λ is the security parameter, l is the length of h in CE and pk in PKE, n is the length of identities in IBE, T and S are respectively the number of identities and the length of mpk of a weakly compact IBE (wcIBE) primitive [8]. The reduction of the modular exponentiations inside garbled circuits leads to the major saving of our scheme. Also, our master public key size grows linearly with the length n of identities in IBE, whereas that in [8] grows linearly with the number T of identities in wcIBE, which should be set large enough to ensure that $S < T/4$.

Sadly, the encryption time and ciphertext size mainly depend on the time to garble circuits and the size of the garbled circuits, respectively, which are hard to compare when the circuits being garbled are different. Roughly, while the number of circuits being garbled is $O(n)$ in both schemes, the complexity of our circuits is lower [8]. Thus, the encryption time and ciphertext size of our scheme shall not become worse [8].

Semantic Security and Blindness. We prove the IND-ID-CPA and IND-BLIND-ID-CPA securities of our scheme. Our construction requires the division of encryption, and thus the blindness property (or some weaker ciphertext divisibility property) of the underlying primitives for even IND-IN-CPA security.

Table 1. Efficiency Comparison of AIBE Schemes

	Decryption time (operations inside circuits)	Size of mpk	Size of sk
Ours	$2\lambda nl + 1$ exponentiations	$O(4nl)$	$O(nl)$
[8]	$4\lambda n/T$ exponentiations	$O(2lT)$	$O(nlTS)$

Theorem 5 (IND-ID-CPA Security). *Our IBE scheme is IND-ID-CPA secure if \mathcal{CE} is a blind CE scheme, \mathcal{PKE} is a blind PKE scheme, \mathcal{GC} is a secure garbling scheme, and \mathcal{F} is a PRF.*

The proof is deferred to Appendix A.2.

Theorem 6 (IND-BLIND-ID-CPA Security). *Our IBE scheme is IND-BLIND-ID-CPA secure if \mathcal{CE} is a blind CE scheme, \mathcal{PKE} is a blind PKE scheme, \mathcal{GC} is a blind garbling scheme, and \mathcal{F} is a PRF.*

The proof is deferred to Appendix A.3.

5 Anonymous Hierarchical Identity-Based Encryption

We present an AHIBE scheme based on the CDH assumption using a blind HIBE scheme via making a one-time signature with encryption (OTSE) construction [18] and, subsequently, an HIBE-from-OTSE construction [15], blind.

5.1 Blind OTSE from Blind CE

An OTSE scheme [15] is a one-time signature scheme that supports encryption and decryption procedures. We define OTSE with selective security properties, which suffices to build selective-ID blind HIBE and adaptive-ID blind IBE.

Definition 15 (Blind One-Time Signature with Encryption). *A blind OTSE scheme consists of five PPT algorithms (Setup, KeyGen, Sign, Enc, Dec):*

- $\text{Setup}(1^\lambda, l)$: This algorithm takes as input a security parameter 1^λ and a message length l , and it outputs public parameters pp .
- $\text{KeyGen}(\text{pp})$: This algorithm takes as input public parameters pp , and it outputs a pair of verification and signing keys (vk, sk) .
- $\text{Sign}(\text{pp}, \text{sk}, x)$: This algorithm takes as input public parameters pp , a signing key sk , and a message x , and it outputs a signature σ .
- $\text{Enc}(\text{pp}, (\text{vk}, i, b), m)$: This algorithm takes as input public parameters pp , a verification key vk , an index i , a bit b , and a plaintext m , and it outputs a ciphertext ct .
- $\text{Dec}(\text{pp}, (\text{vk}, \sigma, x), \text{ct})$: This algorithm takes as input public parameters pp , a verification key vk , a signature σ , a message x , and a ciphertext ct , and it outputs a plaintext m .

We require that a blind OTSE scheme satisfies the following three properties:

◦ *Correctness*: With probability 1 over the randomness of (Setup, KeyGen, Sign, Enc, Dec), we have that $\text{Dec}(\text{pp}, (\text{vk}, \sigma, x), \text{Enc}(\text{pp}, (\text{vk}, i, b), m)) = m$ where $\text{pp} \leftarrow \text{Setup}(1^\lambda, l)$, $(\text{vk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$, $\sigma \leftarrow \text{Sign}(\text{pp}, \text{sk}, x)$ and $x_i = b$.

◦ *Selective Security*: Any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:

1. $\text{pp} \leftarrow \text{Setup}(\lambda, l)$
2. $x \leftarrow \mathcal{A}(\text{pp})$
3. $(\text{vk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$
4. $\sigma \leftarrow \text{Sign}(\text{pp}, \text{sk}, x)$
5. $(i, m_0, m_1, \text{st}) \leftarrow \mathcal{A}(\text{pp}, \text{vk}, \sigma)$
6. $\zeta \leftarrow \{0, 1\}$, $\text{ct} \leftarrow \text{Enc}(\text{pp}, (\text{vk}, i, 1 - x_i), m_\zeta)$
7. $\zeta' \leftarrow \mathcal{A}(\text{st}, \text{ct})$. \mathcal{A} wins if $\zeta' = \zeta$.

◦ *sel-IND-BLIND Security*: there exists some decomposition of $\text{Enc}(\text{pp}, (\text{vk}, i, b), m; r) = \text{E}_1(\text{pp}, i; r) \parallel \text{E}_2(\text{pp}, (\text{vk}, i, b), m; r)$ such that any PPT adversary \mathcal{A} cannot win the following security game with probability greater than $1/2 + \text{negl}(\lambda)$:

1. $\text{pp} \leftarrow \text{Setup}(\lambda, l)$
2. $x \leftarrow \mathcal{A}(\text{pp})$
3. $(\text{vk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$
4. $\sigma \leftarrow \text{Sign}(\text{pp}, \text{sk}, x)$
5. $(i, j, \text{st}) \leftarrow \mathcal{A}(\text{pp}, \text{vk}, \sigma)$
6. $m \leftarrow \mathcal{M}$, $r \leftarrow \mathcal{R}$ (\mathcal{M} and \mathcal{R} are the plaintext and randomness spaces)
7. $\bar{\text{ct}}_1 \leftarrow \text{E}_1(\text{pp}, i; r)$, $\bar{\text{ct}}_2 \leftarrow \text{E}_2(\text{pp}, (\text{vk}, i, j), m; r)$
8. $L = \lceil \bar{\text{ct}}_2 \rceil$, $\bar{\text{ct}}'_2 \leftarrow \{0, 1\}^L$, $\zeta \leftarrow \{0, 1\}$
9. if $\zeta = 0$, then $\text{ct} = (\bar{\text{ct}}_1, \bar{\text{ct}}_2)$; if $\zeta = 1$, then $\text{ct} = (\bar{\text{ct}}_1, \bar{\text{ct}}'_2)$
10. $\zeta' \leftarrow \mathcal{A}(\text{st}, \text{ct})$. \mathcal{A} wins if $\zeta' = \zeta$.

We first construct a blind non-compact OTSE scheme, in which the size of the verification key may be bigger than the size of the messages allowed to be signed. Our construction is based on a blind PKE scheme $\mathcal{PK}\mathcal{E}$ and the non-blind non-compact OTSE construction [18], with modifications highlighted in red.

- $\text{Setup}(1^\lambda, l)$:
 - Compute $\mathcal{PK}\mathcal{E}.\text{pp} \leftarrow \mathcal{PK}\mathcal{E}.\text{Init}(1^\lambda)$.
 - Output $\text{pp} = (\mathcal{PK}\mathcal{E}.\text{pp}, 1^\lambda, l)$.
- $\text{KeyGen}(\text{pp}; r')$:
 - For $j = \{1, \dots, l\}$, $b \in \{0, 1\}$, compute $(\text{pk}_{j,b}, \text{sk}_{j,b}) \leftarrow \mathcal{PK}\mathcal{E}.\text{KeyGen}(1^\lambda)$.
 - Set $\text{vk} \leftarrow \{\text{pk}_{j,0}, \text{pk}_{j,1}\}_{j \in [l]}$ and $\text{sk} \leftarrow \{\text{sk}_{j,0}, \text{sk}_{j,1}\}_{j \in [l]}$.
 - Output (vk, sk) .
- $\text{Sign}(\text{pp}, \text{sk}, x)$: Output $\sigma \leftarrow \{\text{sk}_{j,x_j}\}_{j \in [l]}$.
- $\text{Enc}(\text{pp}, (\text{vk}, i, b), m; \rho)$:
 - Compute $\text{ct}_1 \leftarrow \mathcal{PK}\mathcal{E}.\text{E}_1(\mathcal{PK}\mathcal{E}.\text{pp}; \rho)$.
 - Compute $\text{ct}_2 \leftarrow \mathcal{PK}\mathcal{E}.\text{E}_2(\mathcal{PK}\mathcal{E}.\text{pp}, \text{pk}_{i,b}, m; \rho)$.
 - Output $\text{ct} = (i, \text{ct}_1, \text{ct}_2)$.
- $\text{Dec}(\text{pp}, (\text{vk}, \sigma, x), \text{ct})$: Output $m \leftarrow \mathcal{PK}\mathcal{E}.\text{Dec}(\mathcal{PK}\mathcal{E}.\text{pp}, \text{sk}_{i,x_i}, \text{ct}_1 \parallel \text{ct}_2)$.

Theorem 7. *The above construction is a blind non-compact OTSE construction if $\mathcal{PK}\mathcal{E}$ is blind.*

The proof is easy and thus omitted.

We then construct a blind compact OTSE scheme from a blind non-compact OTSE scheme \mathcal{NC} , a blind garbling scheme \mathcal{GC} , and a blind CE scheme \mathcal{CE} (can as well be a hash encryption since collision tractability is not needed here), basing on the non-blind OTSE construction [18], with modifications highlighted in red.

- $\text{Setup}(1^\lambda, l)$: Compute $\mathcal{NC.pp} \leftarrow \mathcal{NC.Setup}(1^\lambda, l)$, $k \leftarrow \mathcal{CE.KeyGen}(1^\lambda, l')$ (where l' is the size of the verification keys vk). Output $\text{pp} = (\mathcal{NC.pp}, k)$.
- $\text{KeyGen}(\text{pp}; r')$: Compute $(\mathcal{NC.vk}, \mathcal{NC.sk}) \leftarrow \mathcal{NC.KeyGen}(\mathcal{NC.pp})$, and $h \leftarrow \mathcal{CE.H}(k, \mathcal{NC.vk})$. Set $\text{vk} = h$, $\text{sk} = (\mathcal{NC.vk}, \mathcal{NC.sk})$. Output (vk, sk) .
- $\text{Sign}(\text{pp}, \text{sk}, x)$: Compute $\mathcal{NC}\sigma \leftarrow \mathcal{NC.Sign}(\mathcal{NC.pp}, \mathcal{NC.sk}, x)$. Output $\sigma = (\mathcal{NC.vk}, \mathcal{NC}\sigma)$.
- $\text{Enc}(\text{pp}, (\text{vk}, i, b), m; \rho, \{\rho_{j,b'}\}_{j \in [l'], b' \in \{0,1\}})$:
 - Let \mathbb{C} be the following circuit.
 $\mathbb{C}[\mathcal{NC.pp}, i, b, m, \rho](\mathcal{NC.vk})$:
 Compute and output $\mathcal{NC.E}_2(\mathcal{NC.pp}, (\mathcal{NC.vk}, i, b), m; \rho)$.
 - Compute $\text{ct}' \leftarrow \mathcal{NC.E}_1(\mathcal{NC.pp}, i; \rho)$.
 - Compute $(\tilde{\mathbb{C}}, e_{\mathbb{C}}) \leftarrow \mathcal{GC.Garble}(1^\lambda, \mathbb{C}[\mathcal{NC.pp}, i, b, m, \rho])$.
 - Parse $e_{\mathbb{C}} = \{Y_{j,0}, Y_{j,1}\}_{j \in [l']}$.
 - For $j \in [l']$, $b' \in \{0, 1\}$, compute $\text{ct}'_{j,b'} \leftarrow \mathcal{CE.E}_1(k, j; \rho_{j,b'})$ and $\text{ct}''_{j,b'} \leftarrow \mathcal{CE.E}_2(k, (h, j, b'), Y_{j,b'}; \rho_{j,b'})$.
 - Output $\text{ct} = (\tilde{\mathbb{C}}, \text{ct}', \{\text{ct}'_{j,b'}, \text{ct}''_{j,b'}\}_{j,b'})$.
- $\text{Dec}(\text{pp}, (\text{vk}, \sigma, x), \text{ct})$:
 - Let $y = \mathcal{NC.vk}$, compute $\tilde{y} \leftarrow \{\mathcal{CE.Dec}(k, y, \text{ct}'_{j,y_j} \parallel \text{ct}''_{j,y_j})\}_{j \in [l']}$.
 - Compute $c' \leftarrow \mathcal{GC.Eval}(\tilde{\mathbb{C}}, \tilde{y})$.
 - Output $m \leftarrow \mathcal{NC.Dec}(\mathcal{NC.pp}, (y, \mathcal{NC}\sigma, x), \text{ct}' \parallel c')$.

Theorem 8. *Our OTSE construction is selectively secure if \mathcal{CE} is a blind CE scheme, \mathcal{NC} is a non-compact blind OTSE scheme, and \mathcal{GC} is a garbling scheme.*

The proof is completely analogous to that of [18, Theorem 6] and thus omitted. The core idea is that no PPT adversary can distinguish the real security game from a security game in which the challenge ciphertext is computed using the simulator algorithm of the underlying garbling scheme.

Theorem 9. *Our OTSE scheme is sel-IND-BLIND secure if \mathcal{CE} is a blind CE scheme, \mathcal{NC} is a non-compact blind OTSE scheme, and \mathcal{GC} is a blind garbling scheme.*

The proof is deferred to Appendix A.4.

5.2 Blind HIBE from Blind OTSE

We propose the definition for blind HIBE. In blind HIBE, an encryption of a random plaintext is (partially) indistinguishable from random, even when the corresponding secret key is known.

Definition 16 ((Selective-ID) Blind HIBE). *An HIBE scheme is blind if it satisfies the following stronger notion of security:*

◦ *sel-IND-BLIND-ID-CPA Security: There exists some decomposition of $\text{Enc}(\text{mpk}, \text{id}, m; \rho) = E_1(\text{mpk}, |\text{id}|; \rho) \parallel E_2(\text{mpk}, \text{id}, m; \rho)$ such that any PPT adversary \mathcal{A} cannot win the security game below with probability greater than $1/2 + \text{negl}(\lambda)$:*

1. $(\text{id}^*, \text{st}) \leftarrow \mathcal{A}(1^\lambda)$
2. $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$
3. $m \xleftarrow{\$} \mathcal{M}, \rho \xleftarrow{\$} \mathcal{R}$ (\mathcal{M} and \mathcal{R} are the plaintext and randomness spaces)
4. $\bar{\text{ct}}_1 = E_1(\text{mpk}, |\text{id}^*|; \rho), \bar{\text{ct}}_2 = E_2(\text{mpk}, \text{id}^*, m; \rho)$
5. $L = |\bar{\text{ct}}_2|, \bar{\text{ct}}'_2 \xleftarrow{\$} \{0, 1\}^L, \zeta \xleftarrow{\$} \{0, 1\}$
6. if $\zeta = 0$, then $\text{ct} = (\bar{\text{ct}}_1, \bar{\text{ct}}_2)$; if $\zeta = 1$, then $\text{ct} = (\bar{\text{ct}}_1, \bar{\text{ct}}'_2)$
7. $\zeta' \leftarrow \mathcal{A}^{\text{KeyGen}(\text{mpk}, \text{msk}, \cdot), \text{Delegate}(\text{mpk}, \text{msk}, \cdot)}(\text{st}, \text{ct})$.
8. \mathcal{A} wins if $\zeta' = \zeta$ and \mathcal{A} never queried id to its Delegate oracle for any id that is a prefix of id^* .

Theorem 10. *A blind HIBE scheme is also an AHIBE scheme.*

The proof is deferred to Appendix A.5.

We construct a blind HIBE scheme from our blind OTSE scheme, basing on the non-blind HIBE-from-OTSE construction of Döttling and Garg [15], with modifications highlighted in red. Let \mathcal{OTSE} be a blind compact OTSE scheme, and \mathcal{PKE} be a blind PKE scheme, where the verification keys of \mathcal{OTSE} and the public keys of \mathcal{PKE} have the same length l . Let \mathcal{GC} be a blind garbling scheme and \mathcal{F} be a delegatable PRF. We assume for convenience that \mathcal{F} has two output registers \mathcal{F}_1 and \mathcal{F}_2 .

We first define two functions NodeGen, LeafGen, and three circuits T, Q_{last}, Q:

- NodeGen($\text{pp}_1, \text{pp}_2, v, s$):
 - Compute $(\text{vk}_v, \text{sk}_v) = \mathcal{OTSE}.\text{KeyGen}(\text{pp}_1; \mathcal{F}_1(s, v))$.
 - Compute $(\text{vk}_{v\parallel 0}, \text{sk}_{v\parallel 0}) = \mathcal{OTSE}.\text{KeyGen}(\text{pp}_1; \mathcal{F}_1(s, v\parallel 0))$.
 - Compute $(\text{vk}_{v\parallel 1}, \text{sk}_{v\parallel 1}) = \mathcal{OTSE}.\text{KeyGen}(\text{pp}_1; \mathcal{F}_1(s, v\parallel 1))$.
 - Compute $(\text{lpk}_v, \text{lsk}_v) = \mathcal{PKE}.\text{KeyGen}(\text{pp}_2; \mathcal{F}_2(s, v))$.
 - Compute $x_v = \text{vk}_{v\parallel 0} \parallel \text{vk}_{v\parallel 1} \parallel \text{lpk}_v$.
 - Compute $\sigma_v = \mathcal{OTSE}.\text{Sign}(\text{pp}_1, \text{sk}_v, x_v)$.
 - Output $(\text{vk}_v, \sigma_v, x_v)$.
- NodeGen'(pp, s, v): Compute $(\text{lpk}_v, \text{lsk}_v) = \mathcal{PKE}.\text{KeyGen}(\text{pp}, \mathcal{F}_2(s, v))$ and output lsk_v .
- T[pp, m, ρ](lpk): Compute and output $\mathcal{PKE}.E_2(\text{pp}, \text{lpk}, m; \rho)$.

– $Q_{\text{last}}[\text{pp}, \bar{Y}, \bar{\rho}'](\text{vk})$: Compute and output

$$\{\mathcal{OTSE.E}_2(\text{pp}, (\text{vk}, j + 2l, b), Y_{j,b}; \rho_{j,b})\}_{j \in [l], b \in \{0,1\}}.$$

– $Q[\text{pp}, \beta \in \{0, 1\}, \bar{Y}, \bar{\rho}](\text{vk})$: Compute and output

$$\{\mathcal{OTSE.E}_2(\text{pp}, (\text{vk}, j + \beta l, b), Y_{j,b}; \rho_{j+\beta l, b})\}_{j \in [l], b \in \{0,1\}}.$$

Below presents our HIBE scheme (Setup, Delegate, KeyGen, Enc, Dec).

– Setup(1^λ):

- Compute $\mathcal{OTSE.pp} \leftarrow \mathcal{OTSE.Setup}(1^\lambda, 3l)$.
- Compute $\mathcal{PKE.pp} \leftarrow \mathcal{PKE.Init}(1^\lambda)$.
- Choose a random seed s for \mathcal{F} .
- Let $v_0 = \epsilon$ be the root node, compute $(\text{vk}_0, \cdot, \cdot) = \text{NodeGen}(\mathcal{OTSE.pp}, \mathcal{PKE.pp}, v_0, s)$
- Output $\text{mpk} = (\mathcal{OTSE.pp}, \mathcal{PKE.pp}, \text{vk}_0)$ and $\text{msk} = \text{dk}_\epsilon = (s, \emptyset, \perp)$.

– Delegate($\text{mpk}, \text{dk}_{\text{id}}, \text{id}'$):

- Let $n = |\text{id}|$, $n' = |\text{id}'|$, parse $\text{dk}_{\text{id}} = (s_{\text{id}}, \{\text{lk}_i\}_{i \in [n]}, \text{lsk}_{\text{id}})$.
- For $i \in [0, n']$, compute $\text{lk}_{n+i} = \text{NodeGen}(\mathcal{OTSE.pp}, \mathcal{PKE.pp}, \text{id}'[\leq i], s_{\text{id}})$.
- For $i = n'$, compute $\text{lsk}_{\text{id} \parallel \text{id}'} = \text{NodeGen}'(\mathcal{PKE.pp}, \text{id}'[\leq i], s_{\text{id}})$.
- Compute $s_{\text{id} \parallel \text{id}'} = \mathcal{F}.Delegate(s_{\text{id}}, \text{id}')$.
- Output $\text{dk}_{\text{id} \parallel \text{id}'} = (s_{\text{id} \parallel \text{id}'}, \{\text{lk}_i\}_{i \in [n+n']}, \text{lsk}_{\text{id} \parallel \text{id}'})$.

– KeyGen($\text{mpk}, \text{dk}_{\text{id}}, \text{id}'$):

- Compute $\text{dk}_{\text{id} \parallel \text{id}'} = (s, \{\text{lk}_i\}_i, \text{lsk})$ using Delegate($\text{mpk}, \text{dk}_{\text{id}}, \text{id}'$).
- Output $\text{sk}_{\text{id} \parallel \text{id}'} = (\{\text{lk}_i\}_i, \text{lsk})$.

– Enc($\text{mpk}, \text{id}, m; \rho'', \bar{\rho}', \bar{\rho}^{(n-1)}, \dots, \bar{\rho}^{(0)}$):

- Sample randomnesses $\rho'', \bar{\rho}', \bar{\rho}^{(n-1)}, \dots, \bar{\rho}^{(0)}$ where $\bar{\rho}' = (\rho'_{j,b})_{j \in [l], b \in \{0,1\}}$ and $\bar{\rho}^{(i)} = (\rho_{j,b}^{(i)})_{j \in [2l], b \in \{0,1\}}$.
- Compute $(\tilde{T}, \tilde{Y}^T) \leftarrow \mathcal{GC}.Garble(1^\lambda, T[\mathcal{PKE.pp}, m, \rho''])$.
- Compute $\text{ct}'' = \mathcal{PKE.E}_1(\mathcal{PKE.pp}; \rho'')$.
- Let $n = |\text{id}|$, for $i = n, \dots, 0$, proceed as follows:
 - * If $i = n$, derive $(\tilde{Q}^{(n)}, \tilde{Y}^{(n)}) \leftarrow \mathcal{GC}.Garble(1^\lambda, Q_{\text{last}}[\mathcal{OTSE.pp}, \tilde{Y}^T, \bar{\rho}'])$ and $\text{ct}^{(n)} = (\text{ct}_{j,b}^{(n)}) = (\mathcal{OTSE.E}_1(\mathcal{OTSE.pp}, j + 2l; \rho'_{j,b}))_{j \in [l], b \in \{0,1\}}$.
 - * Else, compute $(\tilde{Q}^{(i)}, \tilde{Y}^{(i)}) \leftarrow \mathcal{GC}.Garble(1^\lambda, Q[\mathcal{OTSE.pp}, \text{id}_{i+1}, \tilde{Y}^{(i+1)}, \bar{\rho}^{(i+1)}])$ $\text{ct}^{(i)} = (\text{ct}_{j,b}^{(i)}) = (\mathcal{OTSE.E}_1(\mathcal{OTSE.pp}, j; \rho_{j,b}^{(i)}))_{j \in [2l], b \in \{0,1\}}$.
- Let $y = \text{vk}_{v_0}$ and compute $\tilde{Y}^{(0)} = (Y_{j,y_j}^{(0)})_j$.
- Output $\text{ct} = (\text{ct}^{(0)}, \dots, \text{ct}^{(n)}, \text{ct}'', \tilde{Q}^{(0)}, \dots, \tilde{Q}^{(n)}, \tilde{T}, \tilde{Y}^{(0)})$.

– Dec($\text{mpk}, \text{sk}_{\text{id}}, \text{ct}$):

- Let $n = |\text{id}|$, for $i = 0, \dots, n-1$, proceed as follows:
 - * Compute $\bar{\text{ct}}^{(i)'} \leftarrow \mathcal{GC}.Eval(\tilde{Q}^{(i)}, \tilde{Y}^{(i)})$, parse $\bar{\text{ct}}^{(i)'} = (\text{ct}_{j,b}^{(i)'})_{j \in [l], b \in \{0,1\}}$.

- * Compute $\tilde{y}_j^{(i+1)} = \mathcal{OTSE}.Dec(\mathcal{OTSE}.pp, lk_i, ct_{j',x_i[j']}^{(i)} || ct_{j,x_i[j']}^{(i)'})$, for $j \in [l]$, where $j' = j + id[i + 1] \cdot l$.
- * Let $\tilde{Y}^{(i+1)} = (\tilde{y}_j^{(i+1)})_{j \in [l]}$.
- Compute $\tilde{ct}^{(n)'} \leftarrow \mathcal{GC}.Eval(\tilde{Q}^{(n)}, \tilde{Y}^{(n)})$. Parse $\tilde{ct}^{(n)'} = (c_{j,b}^{(n)})_{j \in [l], b \in \{0,1\}}$.
- Recover $\tilde{Y}^{(n+1)} = (\mathcal{OTSE}.Dec(\mathcal{OTSE}.pp, lk_n, ct_{j,x_n[j+2l]}^{(n)} || ct_{j,x_n[j+2l]}^{(n)'})_{j \in [l]}$ and $ct' \leftarrow \mathcal{GC}.Eval(\tilde{T}, \tilde{Y}^{(n+1)})$.
- Output $m = \mathcal{PKE}.Dec(\mathcal{PKE}.pp, lsk_{id}, ct'' || ct')$.

Table 2. Efficiency Comparison of OTSE Schemes

	Decryption time (operations inside circuits)	Size of mpk	Size of vk	Size of sk
Ours	$4l$ exponentiations	$O(2l^2)$	a string of length l	$4l$ strings of length l
[8]	$2l$ exponentiations	$O(l)$	$2\lambda + 1$ group elements and a string of length l	$2\lambda + 1$ integers

Efficiency. As there is no prior AHIBE scheme from CDH, we compare the efficiency of our scheme with the non-anonymous HIBE-from-CDH schemes [15, 17]. Like our IBE, our AHIBE-from-OTSE construction shares almost the same efficiency as the HIBE-from-OTSE construction [15]. Moreover, the HIBE-from-CDH construction [15] is identical to the latter one [17] when plugging their OTSE-from-CE construction into their HIBE-from-OTSE construction [15].

So, it suffices to compare our OTSE primitive with the one of Döttling and Garg [15], which is shown in Table 2, where λ is the security parameter, and l is the input length to $\mathcal{OTSE}.Setup$, the length of h in CE, and the length of pk in PKE.

One may see that the performance of our scheme is only slightly worse than the one of Döttling and Garg [15], whereas our scheme achieves blindness. Besides, the encryption time and ciphertext size are the time to garble one circuit and the size of one garbled circuit, respectively. While the complexity of a complete encryption procedure inside circuits is similar, our scheme alleviates the complexity of the circuit by pushing a part of the encryption procedure outside of the circuit, which is not allowed in their OTSE scheme [15].

Theorem 11. *Our HIBE scheme is sel-IND-ID-CPA secure if \mathcal{OTSE} is a compact blind OTSE scheme, \mathcal{PKE} is a blind PKE scheme, and \mathcal{GC} is a secure garbling scheme.*

The proof is completely analogous to that of [15, Theorem 4] and thus omitted. The core idea is that no PPT adversary can distinguish the real security game from a security game in which the challenge ciphertext is computed using the simulator algorithm of the underlying garbling scheme.

Theorem 12. *Our HIBE scheme is sel-IND-BLIND-ID-CPA secure if OTSE is a compact blind OTSE scheme, PKE is a blind PKE scheme, and GC is a blind garbling scheme.*

The proof is deferred to Appendix A.6.

6 A(H)IBE from ϕ -Hiding Assumption

We show how to construct A(H)IBE schemes from the ϕ -hiding assumption. In particular, we construct a weakened version of blind CE, namely, the blind hash encryption (HE) scheme, from the ϕ -hiding assumption. Specifically, (blind) HE is (blind) CE scheme without collision tractability. We omit the formal definition for (blind) HE as it is just (blind) CE that removes the trapdoors and the H^{-1} algorithm. We note that a blind HE scheme from ϕ -hiding can be bootstrapped into blind PKE from ϕ -hiding using the framework in [8, Section 7.5 (Full Version)], which together with the blind HE scheme itself can further be bootstrapped into blind HIBE from ϕ -hiding as in Sect. 5.

We construct our blind HE scheme from the ϕ -hiding assumption, basing on a one-way function with encryption (OWFE) scheme [24, Section 6], with modifications highlighted in red. Note that our construction only requires the IND security and IND-BLIND security properties and hence no longer requires the restriction to the message length n for satisfying the two additional onewayness and smoothness properties [24].

Definition 17 (ϕ -hiding Assumption). *Let $\phi(N) = N \cdot \prod_{p|N} (1 - 1/p)$. Let $\mathbf{Primes}(\lambda)$ denote the set of primes of bit-length λ , and let $\mathbf{RSA}(\lambda) = \{N : N = pq; p, q \in \mathbf{Primes}(\lambda/2); \gcd(p-1, q-1) = 2\}$. For any $e \leq 2^\lambda$, let $\mathbf{RSA}_e(\lambda) = \{N \in \mathbf{RSA}(\lambda) : e \text{ divides } \phi(N)\}$. We say that the ϕ -hiding assumption is hard if for all $\epsilon > 0$, integers e such that $3 < e < 2^{\lambda/4-\epsilon}$ and any PPT adversary \mathcal{A} , $\Pr[\mathcal{A}(N, e) = 1 : N \leftarrow \mathbf{RSA}(\lambda)] - \Pr[\mathcal{A}(N, e) = 1 : N \leftarrow \mathbf{RSA}_e(\lambda)] \leq \text{negl}(\lambda)$.*

Let Ext be a randomness extractor such that the extracted string looks uniformly random as long as the source has high randomness. (One can regard the extractor as a hardcore predicate but with output space being a string of some fixed length instead of a bit.) Let $p_1 = 2, p_2 = 3$, and $e_i = \lceil \log_{p_i} N \rceil, f_i = p_i^{e_i}$ for all i . Our blind HE scheme is described as follows.

- $\text{KeyGen}(1^\lambda, n)$: Set RSA modulus length $t = 5\lambda$, and sample RSA modulus $N \leftarrow \mathbf{RSA}(t)$. Next, sample a generator $g \leftarrow \mathbf{Z}_N^*$, $2n$ (λ -bit) primes $e_{i,b} \leftarrow \mathbf{Primes}(\lambda)$ for $i \in [n], b \in \{0, 1\}$, and elements $d_0, d_1 \leftarrow \mathbf{Z}_N$. Then, sample a seed $s \leftarrow S$ of extractor $\text{Ext}_{\lambda, l}$ and output public parameters $k = (N, s, g, \{e_{i,b}\}_{i,b}, d_0, d_1)$.
- $\text{H}(k, x)$: Output $h = g^{f_1 f_2 (d_0 x + d_1)} \prod_i e_{i, x_i} \pmod N$.
- $\text{Enc}(k, (h, i, b), m; \rho)$: Output $\text{ct} = (i, c = g^\rho, e = \text{Ext}(h^\rho / e_{i,b}, s) \oplus m)$.
- $\text{Dec}(k, x, \text{ct})$: Output $m = \text{Ext}(c^{f_1 f_2 (d_0 x + d_1)} \prod_{j \neq i} e_{j, x_j}, s) \oplus e$.

Efficiency. There is no explicit (H)IBE scheme from ϕ -hiding in the literature. Indeed, the only known way to construct (H)IBE from ϕ -hiding is by plugging the existing OWFE scheme [24] into the non-blind version [18] of our framework in Sect. 5. While the efficiency of our blind framework is close to the non-blind framework [18], it suffices to compare our HE construction with the OWFE construction [24]. It is easy to observe that the efficiency of the two constructions is almost the same.

Theorem 13. *Our scheme above is a blind HE under the ϕ -hiding assumption.*

The proof for the IND security is completely analogous to that of [24, Theorem 6.2] and thus omitted. The IND-BLIND security can be easily shown from the property of the randomness extractor Ext.

7 Discussions

Efficient AIBE from Decisional Diffie–Hellman (DDH). As in the seminal paper [17], we can derive a DDH-version of our CE scheme by removing the Goldreich–Levin hardcore predicate, and further obtain a DDH-version of our AIBE scheme. Notably, DDH leads to CE where the recipient-independent part $E_1(k, i; \rho)$ of a ciphertext can be further divided into two parts: the index i , and the remaining components, which are indistinguishable from those of another recipient-independent ciphertext $E_1(k, j; \rho)$ for random $j \in [n]$. With the stronger property, we are able to reduce the ciphertext size of our AIBE-from-CE construction. Specifically, the Enc algorithm in AIBE no longer needs to generate the whole “fake” recipient-independent CE ciphertexts.

A(H)IBE from Lattice. It is easy to show that the HE-from-LWE scheme in [18, Section 3.2] is blind. We can then obtain an A(H)IBE-from-LWE construction using the technique in Sect. 5.

8 Concluding Remarks

Following the seminal result of Döttling and Garg [17], we show how to construct A(H)IBE schemes by applying the blinding technique of Brakerski *et al.* [8] to non-black-box (H)IBE schemes. It would be interesting to explore more efficient construction of anonymous schemes basing on different assumptions.

A Proof of Security

A.1 Proof of Theorem 3

Proof. Let $\bar{ct}_1 = E_1(k, i; \rho) = (i, c = g^\rho, (c_{j,b'})_{j \in [n] \setminus \{i\}, b' \in \{0,1\}})$, $\bar{ct}_2 = E_2(k, (h, i, b), m; \rho) = m \oplus \text{HardCore}(\frac{h^\rho}{g_{i,b}^\rho})$. It is obvious that \bar{ct}_1 is independent of h, b , and m . As $\text{HardCore}(\frac{h^\rho}{g_{i,b}^\rho})$ is deterministic given \bar{ct}_1 , we have that \bar{ct}_2 is random as long as m is random. It follows that any adversary will have exactly $1/2$ probability of winning the IND-BLIND game. \square

A.2 Proof of Theorem 5

Proof. Suppose \mathcal{A} is an efficient adversary playing the IND-ID-CPA security game. We will show that the advantage of \mathcal{A} is negligible with a sequence of hybrids. Let q be a polynomial upper bound on the runtime of \mathcal{A} , and thus also an upper bound for the number of \mathcal{A} 's key queries.

- G_{real} : This game is the original security game, as shown in Definition 4.
- G_0 : This game is identical to G_{real} except that all pseudorandom function calls are responded to using a truly random function.
- G_τ for $\tau \in [1, n]$: For every τ , this game is identical to G_0 except in how the challenge ciphertext is generated. Recall that the challenge ciphertext contains a sequence of $n + 1$ garbled circuits. In G_τ , we generate the first τ of these garbled circuits using the simulator provided by the garbled circuit construction. More formally, to compute a challenge ciphertext for identity id^* , the first τ garbled circuits are generated as follows:

- For $i = \tau - 1, \dots, 0$, parse $\bar{Y}^{(i+1)} = (Y_{j,b}^{(i+1)})_{j,b}$, compute $(\tilde{Q}^{(i)}, (Y_j^{(i)})_j) \leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \text{BatchEnc}(k_i, h_i, \text{id}^*[i + 1], (Y_{j,h_{i+1}[j]}^{(i+1)}, Y_{j,h_{i+1}[j]}^{(i+1)})_j; \bar{\rho}^{(i+1)}))$, and set $\bar{Y}^{(i)} = (Y_j^{(i)}, Y_j^{(i)})_j$.

We note that we can always generate (h_i, x_i, r_i) for $i \in [0, n - 1]$ locally.

- G_{n+1} : This game is identical to G_n except that we generate the $(n + 1)$ -th garbled circuit using the simulator provided by the garbled circuit construction. More formally, to compute a challenge ciphertext for identity id^* , the $(n + 1)$ -th garbled circuit is generated as follows:
 - For $i = n$, compute

$$(\tilde{T}, (Y_j^{(i)})_j) \leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{PK}\mathcal{E}.\text{E}_2(\text{pp}^{\mathcal{PK}\mathcal{E}}, \text{lpk}_{\text{id}^*}, m_\zeta; \rho')),$$

$$\text{and set } \bar{Y}^{(i)} = (Y_j^{(i)}, Y_j^{(i)})_j.$$

We note that even though the adversary is not allowed to query for sk_{id^*} , we can always generate lpk_{id^*} locally.

- G_{final} : This game is identical to G_{n+1} except that we change the ciphertext $\mathcal{PK}\mathcal{E}.\text{E}_2(\text{pp}^{\mathcal{PK}\mathcal{E}}, \text{lpk}_{\text{id}^*}, m_\zeta; \rho')$ hardwired in the simulated garbling of the circuit T to be $\mathcal{PK}\mathcal{E}.\text{E}_2(\text{pp}^{\mathcal{PK}\mathcal{E}}, \text{lpk}_{\text{id}^*}, 0; \rho')$.

The indistinguishability between G_{real} and G_0 follows directly from the pseudorandomness property of \mathcal{F} . The indistinguishability between G_τ and $G_{\tau+1}$ for $\tau \in [0, n - 1]$ is proved in Lemma 1. The indistinguishability between G_n and G_{n+1} follows from the simulation security of \mathcal{GC} . The indistinguishability between G_{n+1} and G_{final} follows from the IND-CPA security and the blindness security of $\mathcal{PK}\mathcal{E}$. Finally, G_{final} is information-theoretically independent of the message m_ζ , in which \mathcal{A} gains no advantage. \square

Lemma 1. G_τ and $G_{\tau+1}$, $\tau \in [0, n - 1]$, are computationally indistinguishable.

Proof. We describe a sequence of hybrid games.

- $H_{\tau,1}$: This game is identical to G_τ except that we change the generation process of the $(\tau + 1)$ -th garbled circuit

$$(\tilde{Q}^{(\tau)}, \bar{Y}^{(\tau)}) \leftarrow \mathcal{GC}.\text{Garble}(1^\lambda, \mathcal{Q}[k_\tau, \text{id}^*[\tau + 1], \bar{Y}^{(\tau+1)}, \bar{\rho}^{(\tau+1)}]),$$

where $\bar{Y}^{(\tau)} = (Y_{j,b}^{(\tau)})_{j,b}$, to

$$(\tilde{Q}^{(\tau)}, (Y_j^{(\tau)})_j) \leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \text{BatchEnc}(k_\tau, h_\tau, \text{id}^*[\tau + 1], \bar{Y}^{(\tau+1)}; \bar{\rho}^{(\tau+1)})),$$

and set $\bar{Y}^{(\tau)} = (Y_j^{(\tau)}, Y_j^{(\tau)})_j$.

When making the change, we are free to compute h_τ and respond to any key queries as we possess the trapdoors of \mathcal{CE} and secret keys of $\mathcal{PK}\mathcal{E}$.

- $H_{\tau,2}$: This game is identical to $H_{\tau,1}$ except that we change how the values h_v and r_v for $v \in \{0, 1\}^\tau$ are calculated when responding to the adversary's key query. Recall that in G_τ , h_v is computed as $\mathcal{CE}.\text{H}(k_\tau, 0^{2l}; \mathcal{F}(s, v))$, and r_v is computed as $\mathcal{CE}.\text{H}^{-1}(t_\tau, (0^{2l}, \mathcal{F}(s, v)), x_v)$. In $H_{\tau,1}$, we choose r_v uniformly and compute $h_v = \mathcal{CE}.\text{H}(k_\tau, x_v; r_v)$.
- $H_{\tau,3}$: This game is identical to $H_{\tau,2}$ except that we change the generation process of the $(\tau + 1)$ -th garbled circuit:

$$(\tilde{Q}^{(\tau)}, (Y_j^{(\tau)})_j) \leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \text{BatchEnc}(k_\tau, h_\tau, \text{id}^*[\tau + 1], \bar{Y}^{(\tau+1)}; \bar{\rho}^{(\tau+1)})),$$

to $(\tilde{Q}^{(\tau)}, (Y_j^{(\tau)})_j)$

$$\leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \text{BatchEnc}(k_\tau, h_\tau, \text{id}^*[\tau + 1], (Y_{j,h_{\tau+1}[j]}^{(\tau+1)}, Y_{j,h_{\tau+1}[j]}^{(\tau+1)})_j; \bar{\rho}^{(\tau+1)})).$$

When making the change, we do not generate (k_τ, t_τ) by ourselves. Instead, we obtain k_τ from a \mathcal{CE} experiment. The output of the BatchEnc function (i.e., the “correct” set of the recipient-dependent CE encryptions) and the “correct” set of the recipient-independent CE encryptions are returned by the \mathcal{CE} experiment. We note that despite the fact that $2l$ randomnesses are input into the BatchEnc function, only l randomnesses are used to generate the “correct” set of recipient-dependent CE encryptions. Thus the challenger is free to newly generate l randomnesses to compute the “fake” set of the recipient-independent CE encryptions with the new randomnesses. Besides, any key queries can be responded to using the method described in $H_{\tau,2}$.

- $H_{\tau,4}$: This game is identical to $H_{\tau,3}$ except that we calculated h_v and r_v as in the original scheme.

The indistinguishability of hybrids G_τ and $H_{\tau,1}$ follows from the simulation security of \mathcal{GC} . The indistinguishability of hybrids $H_{\tau,1}$ and $H_{\tau,2}$ follows from the trapdoor collision and uniformity properties of \mathcal{CE} . The indistinguishability of hybrids $H_{\tau,2}$ and $H_{\tau,3}$ follows from the IND security and the blindness security of \mathcal{CE} . The indistinguishability of hybrids $H_{\tau,3}$ and $H_{\tau,4}$ follows from the trapdoor collision and uniformity properties of \mathcal{CE} . Finally, $H_{\tau,4}$ is identical to $G_{\tau+1}$. \square

A.3 Proof of Theorem 6

Proof. $\text{Enc}(\text{mpk}, \text{id}, m; (\rho', \bar{\rho}^{(1)}, \dots, \bar{\rho}^{(n)}))$ can be decomposed into E_1 and E_2 : $E_1(\text{mpk}; (\rho', \bar{\rho}^{(1)}, \dots, \bar{\rho}^{(n)})) = (\text{ct}^{(0)}, \dots, \text{ct}^{(n)})$, $E_2(\text{mpk}, \text{id}, m; (\rho', \bar{\rho}^{(1)}, \dots, \bar{\rho}^{(n)})) = (\tilde{Q}^{(0)}, \dots, \tilde{Q}^{(n-1)}, \tilde{T}, \tilde{Y}^{(0)})$. Suppose that \mathcal{A} is an efficient adversary playing the IND-BLIND-ID-CPA security game. Let q be a polynomial upper bound on the runtime of \mathcal{A} , and thus also an upper bound for the number of \mathcal{A} 's key queries. We will show that \mathcal{A} gains a negligible advantage in the IND-BLIND-ID-CPA security game, using a sequence of hybrid games. Note that in the hybrids, we only make changes when $\zeta = 0$, i.e., the challenge ciphertext $\text{ct} = (\bar{\text{ct}}_1, \bar{\text{ct}}_2)$. In particular, we will act as the game challenger and interact with \mathcal{A} .

- G_{real} : This game is the original security game, as shown in Definition 14.
- G_0, \dots, G_{n+1} are defined analogously as in Appendix A.2.
- G'_τ for $\tau \in [0, n]$: This game is identical to G_{n+1} except in how the challenge ciphertext is generated. Recall that in G_{n+1} , we generate the $(i+1)$ -th garbled circuit as $\mathcal{GC}.\text{Sim}(1^\lambda, \cdot) \rightarrow (\tilde{Q}^{(i)}, (Y_j^{(i)})_j)$ and set $\bar{Y}^{(i)} = (Y_j^{(i)}, Y_j^{(i)})_j$. (When $i = n$, the generated garbled circuit is \tilde{T} , here we abuse the notion of $\tilde{Q}^{(i)}$ for convenience.) In this game, to compute a challenge ciphertext for identity id^* , the last $(n+1-\tau)$ garbled circuits are generated as follows:
 - For $i = n$, we replace $(\tilde{T}, (Y_j^{(i)})_j)$ with a uniformly random string of the same length.
 - For $i = n-1, \dots, \tau$, we replace $(\tilde{Q}^{(i)}, (Y_j^{(i)})_j)$ with a uniformly random string of the same length.

The indistinguishability of G'_n and G_{n+1} is proved in Lemma 2. The indistinguishability of $G'_{\tau+1}$ and G'_τ for $\tau \in [0, n-1]$ is proved in Lemma 3. In G'_0 , \mathcal{A} will have no advantage in winning the IND-BLIND-ID-CPA security game. \square

Lemma 2. G_{n+1} and G'_n are computationally indistinguishable.

Proof. We describe a hybrid game:

- H'_{n+1} : This game is identical to G_{n+1} except that we change the ciphertext hardwired in the simulated garbling of the $(n+1)$ -th garbled circuit from

$$(\tilde{T}, (Y_j^{(n)})_j) \leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{PKE}.\text{E}_2(\text{pp}^{\mathcal{PKE}}, \text{lpk}_{\text{id}^*}, m; \rho')),$$

to

$$(\tilde{T}, (Y_j^{(n)})_j) \leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{U}),$$

where \mathcal{U} is sampled uniformly at random from the output space of $\mathcal{PKE}.\text{E}_2(\cdot)$.

The indistinguishability of G_{n+1} and H'_{n+1} follows from the IND-BLIND security of \mathcal{PKE} . The indistinguishability of H'_{n+1} and G'_n follows from the IND-BLIND security of \mathcal{GC} . \square

Lemma 3. $G'_{\tau+1}$ and G'_τ are computationally indistinguishable, $\forall \tau \in [0, n-1]$.

Proof. We describe a sequence of hybrid games.

- $H'_{\tau+1,1}$: This game is identical to $G'_{\tau+1}$ except that we calculate values h_v and r_v for $v \in \{0, 1\}^\tau$ as in $H_{\tau,1}$.
- $H'_{\tau+1,2}$: It is identical to $H'_{\tau+1,1}$ except that we change the ciphertext hardwired in the simulated garbling of the $(\tau+1)$ -th garbled circuit: $(\tilde{Q}^{(\tau)}, (Y_j^{(\tau)})_j)$

$$\leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \text{BatchEnc}(k_\tau, h_\tau, \text{id}^*[\tau+1], (Y_{j,h_{\tau+1}[j]}^{(\tau+1)}, Y_{j,h_{\tau+1}[j]}^{(\tau+1)})_j; \bar{\rho}^{(\tau+1)})),$$

to

$$(\tilde{Q}^{(\tau)}, (Y_j^{(\tau)})_j) \leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{U}),$$

where \mathcal{U} is sampled uniformly at random from the output space of $\text{BatchEnc}()$. We note that when making the change, we do not generate (k_τ, t_τ) by ourselves. Instead, we obtain k_τ from a \mathcal{CE} experiment. Although t_τ is not given to us, we can compute h_τ and respond to any key queries using the method described in $H'_{\tau+1,1}$.

- $H'_{\tau+1,3}$: This game is identical to $G''_{\tau+1,2}$ except that we calculate values h_v and r_v as in the original scheme.
- $H'_{\tau+1,4}$: It is identical to $H'_{\tau+1,3}$ except that we change the generation process of the $(\tau+1)$ -th garbled circuit in the challenge ciphertext. In particular, we set $(\tilde{Q}^{(i)}, (Y_j^{(\tau)})_j)$ as a uniformly random string of the same length.

The indistinguishability of $G'_{\tau+1}$ and $H'_{\tau+1,1}$ follows from the trapdoor collision and uniformity properties of \mathcal{CE} . The indistinguishability of $H'_{\tau+1,1}$ and $H'_{\tau+1,2}$ follows from the IND-BLIND security of \mathcal{CE} . The indistinguishability of $H'_{\tau+1,2}$ and $H'_{\tau+1,3}$ follows from the trapdoor collision and uniformity properties of \mathcal{CE} . The indistinguishability of $H'_{\tau+1,3}$ and $H'_{\tau+1,4}$ for $\tau \in [0, n]$ follows from the IND-BLIND security of \mathcal{GC} . We note that $H'_{\tau+1,4}$ is identical to G'_τ . \square

A.4 Proof of Theorem 9

Proof. $\text{Enc}(\text{pp}, (\text{vk}, i, b), m; r = (\rho, \{\rho_{j,b'}\}_{j \in [l], b' \in \{0,1\}}))$ can be decomposed into two parts: $\text{E}_1(\text{pp}, i; r) = (\text{ct}', \{\text{ct}'_{j,b'}\}_{j,b'})$, $\text{E}_2(\text{pp}, (\text{vk}, i, b), m; r) = (\tilde{C}, \{\text{ct}''_{j,b'}\}_{j,b'})$. Suppose that \mathcal{A} is an efficient adversary playing the sel-IND-BLIND security game. We will show that \mathcal{A} gains a negligible advantage in the sel-IND-BLIND security game, using a sequence of hybrid games. In the hybrids, we only make changes when $\zeta = 0$, i.e., the challenge ciphertext $\text{ct} = (\text{ct}_1, \text{ct}_2)$. In particular, we will act as the game challenger and interact with \mathcal{A} .

- G_{real} : This game is the original security game, as shown in Definition 15.
- G_0 : This game is identical to the game H_2 in [18, Theorem 6]. Specifically, the recipient-dependent part of the challenge ciphertext is generated as $(\tilde{C}, (Y_{j,\mathcal{NC}.\text{vk}_j})_j) \leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{NC}.\text{E}_2(\mathcal{NC}.\text{pp}, (\mathcal{NC}.\text{vk}, i, b), m; \rho))$, $\text{ct}''_{j,b'} \leftarrow \mathcal{CE}.\text{E}_2(k, (h, j, b'), Y_{j,\mathcal{NC}.\text{vk}_j}; \rho_{j,b'})$ for $j \in [l], b' \in \{0, 1\}$.
- G_1 : This game is identical to G_0 except in how the challenge ciphertext is generated. In particular, we compute $(\tilde{C}, (Y_{j,\mathcal{NC}.\text{vk}_j})_j)$ as $\mathcal{GC}.\text{Garble}(1^\lambda, \mathcal{U})$ where \mathcal{U} is sampled uniformly at random from the output space of $\mathcal{NC}.\text{E}_2(\mathcal{NC}.\text{pp}, (h, i, b), m; \rho)$.

- G_2 : This game is identical to G_1 except in how the challenge ciphertext is generated. In particular, we replace $(\tilde{C}, (Y_{j, \mathcal{N}C, vk_j})_j)$ by a uniformly random string of the same length.
- G_3 : This game is identical to G_2 except in how the challenge ciphertext is generated. In particular, we replace $ct''_{j, b'}$ for $j \in [l'], b' \in \{0, 1\}$ by uniformly random strings of the same length.

The indistinguishability of G_{real} and G_0 is proved [18, Theorem 6] and thus omitted. The indistinguishability of G_0 and G_1 follows from the sel-IND-BLIND security of $\mathcal{N}C$. The indistinguishability of G_1 and G_2 follows from the IND-BLIND security of $\mathcal{G}C$. The indistinguishability of G_2 and G_3 follows from the IND-BLIND security of $\mathcal{C}E$. In G_3 , \mathcal{A} will have no advantage in winning the sel-IND-BLIND security game.

A.5 Proof of Theorem 10

Proof. Consider an adversary \mathcal{A} playing the sel-IND-ANON-ID-CPA security game of HIBE; \mathcal{A} is eventually given a challenge $ct \leftarrow \text{Enc}(\text{mpk}, \text{id}_\zeta, m)$, where $(\text{id}_0, \text{id}_1, m)$ are chosen by \mathcal{A} . We note that id_0 and id_1 are restricted to the same length. For each $\zeta \in \{0, 1\}$, it is certainly the case that \mathcal{A} cannot distinguish whether it was given $ct_{\text{id}_\zeta, m} \leftarrow \text{Enc}(\text{mpk}, \text{id}_\zeta, m)$ or $ct_{\text{id}_\zeta, m^*} \leftarrow \text{Enc}(\text{mpk}, \text{id}_\zeta, m^*)$, where $m^* \xleftarrow{\$} \mathcal{M}$; this follows from sel-IND-ID-CPA security of HIBE. Additionally, by sel-IND-BLIND-ID-CPA security of HIBE, \mathcal{A} also cannot distinguish whether it is given $ct_{\text{id}_\zeta, m^*}$ as above or $ct'_{\text{id}_\zeta, m^*} = E_1(\text{mpk}, |\text{id}_\zeta|; \rho) \| \mathcal{U}$ for $\rho \xleftarrow{\$} \mathcal{R}$, $\mathcal{U} \xleftarrow{\$} \{0, 1\}^{|\mathcal{E}_2(\text{mpk}, \text{id}_\zeta, m^*; \rho)|}$. As ct'_{id_0, m^*} and ct'_{id_1, m^*} are drawn from identical distributions, we conclude that \mathcal{A} cannot distinguish whether it is given $ct_{\text{id}_0, m}$ or $ct_{\text{id}_1, m}$, as desired. \square

A.6 Proof of Theorem 12

Proof. The encryption algorithm $\text{Enc}(\text{mpk}, \text{id}, m; r = (\rho'', \bar{\rho}', \bar{\rho}^{(n-1)}, \dots, \bar{\rho}^{(0)}))$ of our scheme can be decomposed into two parts: $E_1(\text{mpk}; r) = (ct^{(0)}, \dots, ct^{(n)}, ct'')$, $E_2(\text{mpk}, \text{id}, m; r) = (\tilde{Q}^{(0)}, \dots, \tilde{Q}^{(n)}, \tilde{T}, \tilde{Y}^{(0)})$. Suppose that \mathcal{A} is an efficient adversary playing the sel-IND-BLIND-ID-CPA security game. We will show that \mathcal{A} gains a negligible advantage in the sel-IND-BLIND-ID-CPA security game, using a sequence of hybrid games. We note that in the hybrid games, we only make changes when $\zeta = 0$, i.e., the challenge ciphertext $ct = (\bar{ct}_1, \bar{ct}_2)$. In particular, we will act as the game challenger and interact with \mathcal{A} .

- G_{real} : This game is the original security game, as shown in Definition 16.
- G_0 : It is identical to H_{2n^*+3} in [15, Theorem 4]. Specifically, the PRF function is modified such that all key queries can be responded by the challenger without knowing the trapdoor values $t_v \forall v \in \{\epsilon, \text{id}^*[\leq 1], \dots, \text{id}^*[\leq n-1]\}$. The recipient-dependent part of the challenge ciphertext is generated as:

Compute $(\tilde{T}, (Y_j^T)_j) \leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{PKE}.\text{E}_2(\text{pp}^{\mathcal{PKE}}, \text{pk}_{\text{id}^*}, m; \rho''))$.
 For $i = n, \dots, 0$:
 If $i = n$:
 Compute $(\tilde{Q}^{(n)}, (Y_j^{(n)})_j)$
 $\leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{Q}_{\text{last}}[\mathcal{OTSE}.\text{pp}, (Y_j^T, Y_j^T)_j, \bar{\rho}'](\text{vk}_{\text{id}^*[\leq n]}))$.
 Else:
 Compute $(\tilde{Q}^{(i)}, (Y_j^{(i)})_j)$
 $\leftarrow \mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{Q}[\mathcal{OTSE}.\text{pp}, \text{id}_{i+1}, (Y_j^{(i+1)}, Y_j^{(i+1)})_j, \bar{\rho}^{(i+1)}](\text{vk}_{\text{id}^*[\leq i]}))$.

- $G_{0,2}$: This game is identical to G_0 except in how the challenge ciphertext is generated. In particular, we compute $(\tilde{T}, (Y_j^T)_j)$ as $\mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{U})$, where \mathcal{U} is sampled uniformly at random from the output space of $\mathcal{PKE}.\text{E}_2(\cdot)$.
- $G_{0,1}$: This game is identical to $G_{0,2}$ except in how the challenge ciphertext is generated. In particular, we replace $(\tilde{T}, (Y_j^T)_j)$ by a uniformly random string of the same length.
- $G_{\tau,2}$ for $\tau \in [1, n]$: It is identical to $G_{\tau-1,1}$ except for the challenge ciphertext. Particularly, we compute $(\tilde{Q}^{(n-\tau+1)}, (Y_j^{(n-\tau+1)})_j)$ as $\mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{U}_1)$ if $\tau = 1$ or $\mathcal{GC}.\text{Sim}(1^\lambda, \mathcal{U}_2)$ otherwise, where \mathcal{U}_1 and \mathcal{U}_2 are sampled uniformly at random from the output space of $\mathcal{Q}_{\text{last}}(\cdot)$ and $\mathcal{Q}(\cdot)$ respectively.
- $G_{\tau,1}$ for $\tau \in [1, n]$: This game is identical to $G_{\tau,2}$ except in how the challenge ciphertext is generated. In particular, we replace $(\tilde{Q}^{(n-\tau+1)}, (Y_j^{(n-\tau+1)})_j)$ by a uniformly random string of the same length.

The indistinguishability of G_{real} and G_0 is proved in [15, Theorem 4] and thus omitted here. The indistinguishability of G_0 and $G_{0,2}$ follows from the IND-BLIND security of \mathcal{PKE} . The indistinguishability of $G_{\tau,2}$ and $G'_{\tau,1}$ for $\tau \in [0, n]$ follows from the IND-BLIND security of \mathcal{GC} . The indistinguishability of $G_{\tau-1,1}$ and $G'_{\tau,2}$ for $\tau \in [1, n]$ follows from the sel-IND-BLIND security of \mathcal{OTSE} . In $G_{n,1}$, \mathcal{A} will have no advantage in winning the sel-IND-BLIND-ID-CPA security game. We note that in all of the above games, the challenger is free to answer the key queries with the modified PRF. □

References

1. Abdalla, M., et al.: Searchable encryption revisited: consistency properties, relation to anonymous IBE, and extensions. *J. Cryptol.* **21**(3), 350–391 (2007). <https://doi.org/10.1007/s00145-007-9006-6>
2. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: EUROCRYPT (2010)
3. Boneh, D., Boyen, X.: Efficient selective-ID secure identity-based encryption without random oracles. In: EUROCRYPT (2004)
4. Boneh, D., Crescenzo, G.D., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: EUROCRYPT (2004)
5. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: CRYPTO (2001)

6. Boneh, D., Papakonstantinou, P.A., Rackoff, C., Vahlis, Y., Waters, B.: On the impossibility of basing identity based encryption on trapdoor permutations. In: FOCS (2018)
7. Boyen, X., Waters, B.: Anonymous hierarchical identity-based encryption (without random oracles). In: CRYPTO (2006)
8. Brakerski, Z., Lombardi, A., Segev, G., Vaikuntanathan, V.: Anonymous IBE, leakage resilience and circular security from new assumptions. In: EUROCRYPT (2018)
9. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: EUROCRYPT (1999)
10. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: EUROCRYPT (2003)
11. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. *J. Cryptol.* **25**(4), 601–639 (2011). <https://doi.org/10.1007/s00145-011-9105-2>
12. Cho, C., Döttling, N., Garg, S., Gupta, D., Miao, P., Polychroniadou, A.: Laconic oblivious transfer and its applications. In: CRYPTO Part II (2017)
13. Chow, S.S.M.: Removing escrow from identity-based encryption. In: PKC (2009)
14. Chow, S.S.M., Roth, V., Rieffel, E.G.: General certificateless encryption and timed-release encryption. In: SCN (2008)
15. Döttling, N., Garg, S.: From selective IBE to full IBE and selective HIBE. In: TCC Part I (2017)
16. Döttling, N., Garg, S.: Identity-based encryption from the Diffie-Hellman assumption. In: CRYPTO Part I (2017)
17. Döttling, N., Garg, S.: Identity-based encryption from the Diffie-Hellman assumption. *J. ACM* **68**, 1–46 (2021)
18. Döttling, N., Garg, S., Hajiabadi, M., Masny, D.: New constructions of identity-based and key-dependent message secure encryption schemes. In: PKC Part I (2018)
19. Garg, S., Hajiabadi, M.: Trapdoor functions from the computational Diffie-Hellman assumption. In: CRYPTO Part II (2018)
20. Garg, S., Hajiabadi, M., Mahmood, M., Rahimi, A.: Registration-based encryption: removing private-key generator from IBE. In: TCC Part I (2018)
21. Gentry, C.: Practical identity-based encryption without random oracles. In: EUROCRYPT (2006)
22. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: FOCS (1984)
23. Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: STOC (1989)
24. Goyal, R., Vusirikala, S., Waters, B.: New constructions of hinting PRGs, OWFs with encryption, and more. In: CRYPTO Part I (2020)
25. Koppula, V., Waters, B.: Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption. In: CRYPTO Part II (2019)
26. Papakonstantinou, P.A., Rackoff, C., Vahlis, Y.: How powerful are the DDH hard groups? *IACR Cryptol. ePrint Arch.* 2012/653 (2012)
27. Waters, B.: Efficient identity-based encryption without random oracles. In: EUROCRYPT (2005)
28. Waters, B.: Dual system encryption: realizing fully secure IBE and HIBE under simple assumptions. In: CRYPTO (2009)



Publicly Auditable Functional Encryption

Vlasis Koutsos^(✉) and Dimitrios Papadopoulos

The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong
{vkoutsos,dipapado}@cse.ust.hk

Abstract. We introduce the notion of *publicly auditable functional encryption* (PAFE). Compared to standard functional encryption, PAFE operates in an extended setting that includes an entity called *auditor*, besides key-generating authority, encryptor, and decryptor. The auditor requests function outputs from the decryptor and wishes to check their correctness with respect to the ciphertexts produced by the encryptor, without having access to the *functional secret key* that is used for decryption. This is in contrast with previous approaches for result verifiability and consistency in functional encryption that aim to ensure decryptors about the legitimacy of the results they decrypt.

We propose four different flavors of public auditability with respect to different sets of adversarially controlled parties (only decryptor, encryptor-decryptor, authority-decryptor, and authority-encryptor-decryptor) and provide constructions for building corresponding secure PAFE schemes from standard functional encryption, commitment schemes, and non-interactive witness-indistinguishable proof systems. At the core of our constructions lies the notion of a *functional public key*, that works as the public analog of the functional secret key of functional encryption and is used for verification purposes by the auditor. Crucially, in order to ensure that these new keys cannot be used to infer additional information about plaintext values (besides the requested decryptions by the auditor), we propose a new indistinguishability-based security definition for PAFE to accommodate not only functional secret key queries (as in standard functional encryption) but also functional public key and decryption queries. Finally, we propose a publicly auditable multi-input functional encryption scheme (MIFE) that supports inner-product functionalities and is secure against adversarial decryptors. Instantiated with existing MIFE using “El Gamal”-like ciphertexts and Σ -protocols, this gives a lightweight publicly auditable scheme.

Keywords: Functional Encryption · Auditability · Public Verifiability

1 Introduction

Functional Encryption (FE) [17, 31] is a cryptographic primitive that transcends the “all-or-nothing” decryption capabilities that classical public-key encryption schemes provide, by allowing the decryptor to acquire pre-agreed function outputs of the encryptor’s data. In particular, given an *encryption key* ek , the

encryptor produces a ciphertext ct for plaintext data x . The decryptor can then use a *functional secret key* sk_f , for an agreed function f , provided by a key-generation authority, in order to retrieve the function output $f(x)$. The security of FE guarantees that the decryptor learns *nothing* about x other than $f(x)$ (and what can be inferred from it). Since it was first proposed by Boneh et al. [17] there has been a plethora of works on FE e.g., providing efficient schemes for specific functionalities (inner product [8], quadratic [6, 9, 12, 25]), and generalizations of the definition for multiple inputs (Multi-Input FE (MIFE) [24, 36]), multiple encryptors or authorities (Multi-Client FE (MCFE) [5, 21, 29] and Multi-Party FE [7]), or dynamic sets of participants (Decentralized Dynamic FE [20]).

The selective decryption capabilities that FE provides makes it suitable for multiple real-world applications, such as the following:

Checkable Cloud Storage. Consider an application where users wish to upload their files (e.g., images) onto a cloud service provider. For privacy purposes uploaded files should be encrypted. However, from the cloud service provider’s perspective it would be ideal if it could still infer some information about the encrypted files, with the consent of the users. For example, it might wish to learn certain file-related aggregate statistics, or ensure that the files’ contents *satisfy certain policies* (e.g., the uploaded images do not contain states of *déshabillé* [2]). Using FE to produce the ciphertext and providing the cloud server with appropriate sk_f simultaneously achieves the above privacy and utility properties.

Auditable Financial Data. Another example application comes from the field of finance, where there exist institutions (e.g., [1, 3]) that are in charge of examining the books of business-conducting companies and organizations to ensure that they conform to legal and financial policies. Again, FE enables such fine-grained control without necessarily disclosing all raw data.

Private Data Brokerage. The abundance of data produced daily from our online activity has naturally led to the notion of “data monetization”. Companies operating as *data brokers* collect large volumes of data, perform useful statistical analyses over them and market the results to interested buyers (e.g., for marketing or political campaigns [22]). Due to the potentially sensitive nature of the raw data, privacy-aware individuals could opt to use an FE scheme that only reveals to the broker pre-agreed aggregate statistics.

While FE seems to map very nicely to the privacy requirements of these applications, the trust model in the settings described allows for potential misbehavior from parties that is not captured by FE security definitions. Note that in all applications above *the party that plays the role of the FE-decryptor is different than the party that needs to be convinced about the legitimacy of the function outputs*. For example, the cloud service provider may need to prove to a law enforcement agency that the stored encrypted data do not violate any legal

policies and likewise, for our second example, auditing companies may need to provide guarantees to governmental regulatory bodies. Finally, in our last example, data buyers should receive a guarantee about the correctness and consistency of the purchased statistical analyses results with respect to the raw data, to avoid having to blindly trust the broker. Based on the above, FE only partially achieves the security requirements of the mentioned applications (allowing the decryptor to only learn function outputs). Particularly, it needs to be augmented or complemented with other cryptographic techniques to enable results verification.

There exists a limited number of works in the FE literature that aims to enrich schemes with some notion of result verification. Badrinarayanan et al. [11] proposed *verifiable FE* and more recently Badertscher et al. [10] proposed a relaxation of this notion under various adversarial models called *consistent FE*. However, both these works operate in a different security model than the ones described above, as they *aim to protect the decryptor* against attacks launched by a misbehaving encryptor, authority, or both. That is, at a very high level, the decryptor can run a verification algorithm that ensures there is a common “explanation” for all decrypted function outputs (see detailed discussion in related work section below). That said, this verification algorithm may require access to the functional decryption keys sk_f for different functions f , which is perfectly acceptable in case it will be executed by the decryptor (who is already entrusted with these keys). Importantly, the problem we are concerned with is strictly harder: an external party needs to be able to verify decrypted results without access to sk_f and *the decryptor is always assumed to behave adversarially to this goal*. Hence none of these prior works for verifiable FE is applicable.

One “easy” solution to our problem would be to provide the external auditing party with sk_f , allowing it to decrypt results directly. However, access to sk_f allows the decryption of arbitrary sets of ciphertexts (e.g., in an application where ciphertexts are publicly accessible). While this might be acceptable for some cases, a more “controlled disclosure,” is more realistic and preferable in other scenarios i.e., when the auditor has to go through the decryptor first. Finally, an alternative “folklore” way for verifying received results is if one can rely on an active direct communication channel between the encryptor and the auditor. In such a setting, the encryptor can use a generic zero-knowledge proof (ZKP) protocol to convince the auditor, when asked, that its plaintext does not deviate from certain policies. This is far from a realistic assumption though, as the encryptor might be unavailable or even unwilling to provide the auditor with such information. An alternative to this would be for the encryptor to produce, ahead of time and alongside its ciphertext, corresponding proofs of validity, provably adhering to certain policies. Policies change, unfortunately, meaning that the encryptor would need to engage in multiple ZKP executions over time, which is undesirable for the same reasons as the previous example.

Table 1. Auditability versions for general FE depending on the untrusted parties and cryptographic building blocks used for the corresponding PAFE schemes. Legend: \circ =Honest, \bullet =Adversarial. Ad/Sel stands for adaptively/selectively secure.

Sec	Encryptor	Authority	Decryptor	Public Auditability	Security	Building Blocks	
5.1	\circ	\circ	\bullet	PA-UD	Ad	Ad-FE	Com + NIWI
5.1	\bullet	\circ	\bullet	PA-UED	Ad	Ad-FE	Com + NIWI
5.2	\circ	\bullet	\bullet	PA-UAD	Ad	4×Ad-FE	Com + NIWI
5.2	\bullet	\bullet	\bullet	PA-UEAD	Sel	4×Sel-FE	Com + NIWI

1.1 Our Results

Motivated by the above, we introduce the notion of *publicly auditable functional encryption (PAFE)*, which operates in an extended setting that includes, besides the authority, encryptor, and decryptor, an additional party named *auditor*. PAFE enables the auditor to verify the decryption results/functional outputs received from the decryptor. Unlike previous works on FE verification [10, 11], PAFE achieves “public verifiability”, i.e., the auditor can run the verification algorithm *without access to* sk_f . In order to achieve this, we introduce a new public parameter that we term *functional public key* (pk_f) for function f . This is provided by the authority as a public analog of sk_f and it operates as an “anchor-of-trust” for the auditor’s verification.

Next, we define different flavors of public auditability, based on different corrupted sets among authority, encryptor, and decryptor. We stress that public auditability is meaningful as a property when the decryptor is untrusted; it is trivially achieved if the auditor trusts the decryptor-provided results. Besides, any combination of the remaining entities (encryptor, authority, or both) may be colluding with the decryptor to “fool” the auditor. Thus, we propose four definitions of auditability for FE corresponding to different untrusted-participants sets PA-UD, PA-UED, PA-UAD, PA-UEAD corresponding to untrusted decryptor, encryptor-decryptor, authority-decryptor, and encryptor-authority-decryptor, respectively, as shown in Table 1, and we explore the relations among them.

Besides auditability, PAFE must maintain the standard security definition of FE [17, 21] i.e., the decryptor learns nothing about the plaintext, except for function outputs. Additionally, the inclusion of functional public keys enables a broader class of attacks from an adversary that has access *only* to ciphertexts, pk_f , and possibly function outputs y ; but not sk_f . To elevate the security model to a more realistic scenario that includes the cases we explained above, we consider a mixed class of adversaries that may acquire access to functional secret or public keys *in a dynamic way*. Additionally, we provide the adversary with oracle access to the encryption and decryption algorithms. Recall that a secure FE scheme allows for any adversary to win with non-negligible advantage only if for all queried functional secret keys and ciphertexts, all underlying plaintexts pairs have equal function outputs. In our PAFE extended model, adversaries may attempt to abuse the decryption capabilities on a ciphertext ct to infer

functional evaluations for a function f for which acquiring the respective sk_f would violate the FE winning conditions. To capture all adversarial cases, we propose a new security definition for PAFE, extending the one for FE.

Additionally, we present four different constructions, each one satisfying a different public auditability flavors. Table 1 depicts the respective cryptographic components and the achieved Security/Public-Auditability types. As we discuss in the next part, we build our PAFE using secure FE schemes, enhanced with appropriate use of commitment schemes to “anchor” the functional public keys pk_f , and non-interactive witness indistinguishability (NIWI) proof systems to force the untrusted parties to prove the correctness of their computations. For the untrusted key-generating authority case (PA-UAD, PA-UEAD), we also expand upon the replication techniques of Badrinarayanan et al. [11].

Our first four schemes work for arbitrary classes of functions (as long as the underlying FE supports them). In the last part of the paper we design a publicly auditable multi-input FE (MIFE) scheme specifically for inner-product functionalities, i.e., linear combinations between encrypted vectors and public weights. Using the MIFE of [8], which produces “El Gamal”-style ciphertexts as a building block, combined with classic Σ -protocols [14, 18] yields an efficient publicly auditable scheme against untrusted decryptors.

1.2 Overview of Techniques

The introduction of the functional public key in the FE setting poses a “modeling” challenge. Even though pk_f should be uniquely tied to sk_f for a specific function f , it should not reveal anything about its private counterpart or the plaintext of the encryptor. To ensure this, we compute pk_f as a perfectly binding and computationally hiding commitment of sk_f . Due to the perfectly binding property no adversary can generate two different functional secret keys that map to the same functional public key which makes the auditability properties easier to prove. The computational hiding property ensures that no bounded-resources adversary can infer any information about sk_f from its pk_f counterpart, hence function outputs that have not been explicitly queried for via the decryption oracle are protected.

However, augmenting IND-secure FE schemes to build secure PAFE schemes is not trivial, due to the diversification of the winning conditions between the IND-security game for FE schemes and that of PAFE that we mentioned above. One way to achieve this would be to restrict ourselves to weaker adversaries. E.g., consider an extremely limited setting where the adversary (after setup) first declares all the secret and public functional keys it wants, as well as all plaintext pairs to be encrypted and ciphertexts to be decrypted. Having access to all this information allows us to check which winning conditions are not violated each time and construct the keys accordingly (either “honestly” or as “dummies”). Clearly, we would prefer to achieve security against more general adversarial behavior. To this end, we use NIWI proofs to combat such adversarial behavior based on the different winning conditions of the two games and achieve fully *adaptive* security, i.e., there is no restriction in the order in which

the adversary issues its different queries. Specifically, our PA-UD and PA-UED constructions leverage the setup performed by the trusted authority to construct an adaptively secure PAFE scheme based on an adaptively secure FE scheme, perfectly binding and computationally hiding commitments and perfectly sound and computationally witness indistinguishable NIWI proof system.

For the cases where the authority is considered untrusted, we need to employ multiple instances of FE where the function output derives from the *majority* of the decryptions and enrich our NIWI relations accordingly, using techniques similar to [11]. Importantly, in the PA-UAD setting we can still construct an adaptively secure PAFE scheme by leveraging that the encryptor is trusted. This derives specifically from a combination of all ciphertexts using a common plaintext x with complex relations proving a threshold of correct execution of the Setup, Key-Generation, and Decryption algorithms. However, in the last case where all entities are untrusted PA-UEAD we do not have such guarantees and we can only achieve a *selective* type of security, as follows: We require that the encryptor provide all plaintext pair queries, before generating any functional secret keys. While this is more limited than the adaptive security of the first three schemes, it arguably suffices for certain applications (e.g., for the cloud application we discussed above, consider the case where all data is uploaded ahead of time, before auditing functions are chosen).

The four different corruption sets result in four discrete public auditability definitions of their own, which we divide into two subcategories depending on whether the authority is trusted or not. A similar division can also be done on whether the encryptor is trusted or not, which results in an interesting realization about the public auditability definitions. For the former cases (PA-UD,PA-UAD), the auditor is guaranteed to receive the function output $f(x)$ that corresponds to the plaintext x that the encryptor has used to produce its ciphertext, and specifically for function f that the auditor holds its respective pk_f . However, for the cases where the encryptor is untrusted (PA-UED,PA-UEAD), it can generate its ciphertext arbitrarily. Therefore, a notion of “consistency” has to suffice to the auditor, which results to an existential condition for x to be in the domain of f (whose respective pk_f the auditor holds and decryption returns $f(x)$).

2 Related Work

Badrinarayanan et al. [11] were the first to consider, in the FE setting, the possibility of encryptors colluding with the authority to try and “cheat” the decryptor into receiving falsified or meaningless function outputs. To safeguard against such attacks, they proposed verifiable FE, including algorithms for the verification of the ciphertext production and the key generation. The verifiability property states that if both algorithms succeed, then decryptors always get a function output (for function f) of a plaintext in the domain of f . This is a very strong definition, as it quantifies over all mpk, ct, f, sk_f the existence of a plaintext x whose function outputs are the result of the decryption algorithm. Interestingly, this is actually the best decryptors can ask for, since the ciphertext

can be generated arbitrarily. More efficient verifiable FE schemes were more recently proposed for inner-product functionalities using pairing-based NIWIs and a perfectly correct inner-product functional encryption scheme [34], and for general functionalities via the use of trusted execution environments [35].

Following in the same line of works, Badertscher et al. [10] proposed the notion of consistency for FE schemes which builds on the idea of [11] in the following way. Additionally to considering the case where both the encryptor and the authority collude, they examine also the cases where just one of the two entities try to “cheat” the decryptor. They observe that in both cases where the encryptor is corrupted, a similar property as in [11] suffices, however, when the encryptor is honest, then the decryptor could request a stronger guarantee, i.e., that it gets the function output of the *specific* plaintext x that the encryptor used to generate the ciphertext ct . The authors explore the relations between their consistency notions and other notions of security (i.e., IND-CPA, IND-CCA, CFE), provide compilers to build consistent FE schemes from FE, NIZKs, and NIWIs, and provide concrete constructions for consistent FE for inner-product functionalities, in the presence of a corrupted encryptor.

At first, it might seem that our notion of public auditability can be achieved by the above schemes. However, even though there exist similarities specifically between our constructions and the ones of [10, 11], public auditability aims to protect the auditor who lacks knowledge of sk_f and knows instead only pk_f . In that sense, our approach can be broadly viewed as a public-key analog of the notions of verifiability and consistency. Thus, not only our constructions require additional techniques for auditability, but PAFE requires a modified security definition, expanded to capture additional adversarial cases, as discussed above.

Koutsos et al. [28] proposed a construction of a privacy-preserving data marketplace that uses FE to protect the privacy of the raw data. To the best of our knowledge, this is the only work that considers a similarly augmented setting for FE with auditability and provides a solution without relying on trusted communication between the encryptor and the auditor. Similarly to our work, they utilize a public equivalent of sk_f to ensure the legitimacy of the brokered results against an untrusted auditor (but not untrusted authority/encryptor or any combination between them). Moreover, they only consider “passive” attacks from parties that observe ciphertexts but without decryption capabilities. Finally, their scheme builds on the MCFE scheme of [21] which supports only inner-products functionalities.

Barbosa et al. [13] were concerned with a somewhat similar problem to ours. In that work the authors proposed a cryptographic primitive called Delegatable Homomorphic Encryption (DHE). With DHE schemes a weak client is convinced that the decryption (which was performed by a potentially malicious cloud service provider) was performed correctly. The authors assume a trusted communication channel between the encryptor and the auditor, so that the latter could be convinced about the legitimacy of the results received from decryptors. However, this is a strong assumption and an unrealistic one as well, as the content provider could go offline after the uploading of its data onto the cloud service

$WI_{\beta}^{\text{NIWI}}(1^{\lambda}, \mathcal{A})$ (for relation R)
$(x, w_1, w_2, st) \leftarrow \mathcal{A}_1(1^{\lambda})$
$\pi \leftarrow \text{NIWI.Prove}(1^{\lambda}, x, w_{\beta})$
$\alpha \leftarrow \mathcal{A}_2(\pi, st)$
Output: $\alpha \wedge (x, w_1) \in R \wedge (x, w_2) \in R$

Fig. 1. Witness-indistinguishability game of a NIWI proof system.

provider. Contrary to our work, the authors of [13] do not consider any possible corruptions from the encryptor or the authority and rely on both FE and fully-homomorphic encryption [23] schemes.

3 Preliminaries

Below we present the necessary cryptographic background for our work.

General Notation. We denote by $\mathcal{F} = \{\mathcal{F}_{\lambda}\}_{\lambda \in \mathbb{N}}$ a family of sets \mathcal{F}_{λ} of functions $f : \mathcal{X}_{\lambda} \rightarrow \mathcal{Y}_{\lambda}$. We call \mathcal{F}_{λ} a functionality class such that all functions $f \in \mathcal{F}_{\lambda}$ have the same domain and the same range. A function $\text{negl}(\lambda) : \mathbb{N} \leftarrow \mathbb{R}^{+}$ is negligible if for every positive polynomial $p(\lambda)$ there exists a $\lambda_0 \in \mathbb{N}$, such that for all $\lambda > \lambda_0 : \epsilon(\lambda) < 1/p(\lambda)$. We denote $[n] = \{1, \dots, n\}$ for $n \in \mathbb{N}^{*}$. For algorithms \mathcal{A} and \mathcal{B} , we write $\mathcal{A}^{\mathcal{B}(\cdot)}(x)$ to denote that \mathcal{A} gets x as an input and has oracle access to \mathcal{B} , that is, the response for an oracle query q is $\mathcal{B}(q)$. Oracles increment a counter every time they receive a query and associate the input-output pairs with their counter, so they can answer repeated queries consistently. Last, we denote by $\leftarrow_{\$} D$ sampling uniformly at random from domain D .

NIWI Proof Systems. A *non-interactive witness-indistinguishable proof system (NIWI)* allows a prover to convince a verifier about the validity of the statement in a way that guarantees that “cheating” provers cannot succeed in this. On the other hand, witness indistinguishability means that no verifier interacting with an honest prover can distinguish which of two witnesses w_1, w_2 was used by the latter (assuming such witnesses exist for the statement).

Definition 1. (Non-Interactive Witness-Indistinguishable Proofs [10]) *Let R be an NP relation and consider the language $L = \{x \mid \exists w \text{ with } (x, w) \in R\}$. A non-interactive witness-indistinguishable proof (NIWI) for the relation R is a tuple of PPT algorithms $\text{NIWI} = (\text{NIWI.Prove}, \text{NIWI.Verify})$:*

$\text{NIWI.Prove}(1^{\lambda}, x, w)$: *Takes as input the unary representation of the security parameter λ , a statement x and a witness w , and outputs a proof π .*

$\text{NIWI.Verify}(1^{\lambda}, x, \pi)$: *Takes as input the unary representation of the security parameter λ , a statement x , and a proof π , and outputs 0 or 1.*

A NIWI is complete, if for all statement-witness pairs in the relation $(x, w) \in R$, it holds that: $\Pr[\text{NIWI.Verify}(1^{\lambda}, x, \text{NIWI.Prove}(1^{\lambda}, x, w)) = 1] = 1$. Besides, a NIWI fulfills the properties of soundness and witness-indistinguishability.

(NIWI Soundness). We define the advantage of an adversary \mathcal{A} as:

$$Adv_{NIWI, \mathcal{A}}^{Sound}(\lambda) := \Pr[(x, w) \leftarrow \mathcal{A}(1^\lambda) : NIWI.Verify(x, \pi) = 1 \wedge x \notin L]$$

A NIWI is perfectly sound if $Adv_{NIWI, \mathcal{A}}^{Sound}(\lambda) = 0$ for all algorithms \mathcal{A} , and computationally sound, if $Adv_{NIWI, \mathcal{A}}^{Sound}(\lambda) \leq \text{negl}(\lambda)$ for all PPT algorithms \mathcal{A} .

(Witness-Indistinguishability). For $\beta \in \{0, 1\}$, we define the experiment $WI_\beta^{NIWI}(1^\lambda, \mathcal{A})$ in Fig. 1. The advantage $Adv_{NIWI}^{WI}(\mathcal{A}(1^\lambda))$ of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is:

$$|\Pr[WI_0^{NIWI}(1^\lambda, \mathcal{A}) = 1] - \Pr[WI_1^{NIWI}(1^\lambda, \mathcal{A}) = 1]|$$

A NIWI is witness-indistinguishable, if $Adv_{NIWI}^{WI}(\mathcal{A}(1^\lambda)) = 0$ for all algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, and computationally witness-indistinguishable, if the advantage $Adv_{NIWI}^{WI}(\mathcal{A}(1^\lambda)) \leq \text{negl}(\lambda)$ for all PPT algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. In our PAFE schemes, we use NIWIs with perfect soundness and computational witness indistinguishability, such as the ones proposed in [15]. We also note that for the configurations with trusted PAFE authority (Sect. 5.1), our NIWIs can be readily replaced with zero-knowledge proof systems that provide a stronger flavor of witness privacy.

Commitments. This primitive allows a party to first commit to a message in a way that reveals nothing about it, and later open it in way that guarantees it cannot “equivocate” for multiple openings. We rely on perfectly binding, computationally hiding commitments (e.g., built from one-way permutations [15]).

Definition 2 (Commitment Schemes). A commitment scheme consists of a pair of PPT algorithms (*Com.Setup*, *Com.Commit*). The *Com.Setup* algorithm $\text{pp} \leftarrow \text{Com.Setup}(1^\lambda)$ generates public parameters pp for the scheme, for security parameter λ . The commitment algorithm defines a function $\mathcal{M}_{\text{pp}} \times \mathcal{R}_{\text{pp}} \rightarrow \mathcal{C}_{\text{pp}}$, for message space \mathcal{M}_{pp} , for randomness space \mathcal{R}_{pp} , and for commitment space \mathcal{C}_{pp} , determined by pp . It takes as input a message x and randomness r and outputs $c \leftarrow \text{Com.Commit}_{\text{pp}}(x; r)$. A perfectly binding and computationally hiding commitment scheme must satisfy the following properties:

- **Perfectly Binding:** Two different strings cannot have the same commitment. More formally, $\forall x_0 \neq x_1, r_0, r_1, \text{Com.Commit}(x_0; r_0) \neq \text{Com.Commit}(x_1; r_1)$.
- **Computationally Hiding:** For all strings x_0 and x_1 (of the same length), for all non-uniform PPT adversaries \mathcal{A} , we have that $Adv^{\text{Com-Hiding}}(\mathcal{A})$ is the advantage defined as:

$$|\Pr[\mathcal{A}(\text{Com.Commit}(x_0; r_0))=1] - \Pr[\mathcal{A}(\text{Com.Commit}(x_1; r_1))=1]| < \text{negl}(\lambda)$$

3.1 Functional Encryption

A functional encryption scheme [9, 17] can be used to encrypt a message, akin to “standard” encryption. However, using special function-specific decryption keys, it also allows a decryptor to learn specific function outputs of the message, but nothing else. The following definition is based on [17, 21].

Definition 3 (Functional Encryption). *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of sets \mathcal{F}_λ of functions $f : \mathcal{X}_\lambda \rightarrow \mathcal{Y}_\lambda$. An FE scheme for the functionality class \mathcal{F}_λ is a tuple of four algorithms $\text{FE} = (\text{FE.Setup}, \text{FE.KeyGen}, \text{FE.Enc}, \text{FE.Dec})$.*

- $\text{FE.Setup}(1^\lambda)$: Takes as input the security parameter λ as 1^λ and outputs a master public key mpk , a master secret key msk , and an encryption key ek .
- $\text{FE.KeyGen}(\text{mpk}, \text{msk}, f)$: Takes as input a master public key mpk , a master secret key msk and a function $f \in \mathcal{F}_\lambda$, and outputs a functional secret key sk_f .
- $\text{FE.Enc}(\text{ek}, x)$: Takes as input an encryption key ek and a string $x \in \mathcal{X}_\lambda$, and outputs a ciphertext ct .
- $\text{FE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct})$: Takes as input a master public key mpk , a function $f \in \mathcal{F}_\lambda$, a functional secret key sk_f , and a ciphertext ct . It outputs a function value $y \in \mathcal{Y}_\lambda$ or \perp , indicating an invalid ciphertext.

Correctness. *An FE scheme is correct, if $\forall \lambda \in \mathbb{N}, \forall f \in \mathcal{F}_\lambda, \exists x \in \mathcal{X}_\lambda$ s.t.:*

$$\Pr \left[\text{FE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct}) = f(x) \mid \begin{array}{l} (\text{msk}, \text{mpk}, \text{ek}) \leftarrow \text{FE.Setup}(1^\lambda) \\ \text{sk}_f \leftarrow \text{FE.KeyGen}(\text{mpk}, \text{msk}, f) \\ \text{ct} \leftarrow \text{FE.Enc}(\text{ek}, x) \end{array} \right] > 1 - \text{negl}(\lambda)$$

The security of FE is captured by the following indistinguishability game. At a high level, the adversary has access to a “left-right” encryption oracle and a key generation for functions of its choice. Importantly, the game detects whether the adversary ever submit an encryption query for a pair of messages with different output for some of the queried functionalities (trivially breaking the game).

Definition 4 (IND Security for FE). *Let us consider an FE scheme. No PPT adversary \mathcal{A} should be able to win the following game against a challenger \mathcal{C} :*

- *Initialization:* \mathcal{C} runs the setup algorithm $(\text{mpk}, \text{msk}, \text{ek}) \leftarrow \text{FE.Setup}(1^\lambda)$ and chooses a random bit $b \leftarrow_s \{0, 1\}$. It provides the master public key mpk to \mathcal{A} .
- *Encryption queries* $\text{QEnc}(x_0, x_1)$: \mathcal{A} has unlimited adaptive access to a Left-or-Right encryption oracle and receives ciphertext ct , generated by $\text{FE.Enc}(\text{ek}, x_b)$.
- *Functional key queries* $\text{QKeyGen}(f)$: \mathcal{A} has unlimited and adaptive oracle access to the $\text{FE.KeyGen}(\text{mpk}, \text{msk}, f)$ algorithm for any input function f of its choice. It is given back the functional secret key sk_f .
- *Finalize:* \mathcal{A} provides its guess b' on the bit b and this procedure outputs the result β of the security game, according to the analysis given below.

Experiment $\text{IND}_{\mathcal{A}}^{\text{MIFE}}(1^\lambda)$

$(n, st_0) \leftarrow \mathcal{A}_0(1^\lambda)$

$\{\text{ek}_i\}_{i \in [n]} \leftarrow \text{MIFE.Setup}(1^\lambda, n)$

$(\mathbf{X}^0, \mathbf{X}^1, st_1) \leftarrow \mathcal{A}_1^{\text{MIFE.KeyGen}(\text{mpk}, \text{msk}, \cdot)}(st_0, \{\text{ek}_i\}_{i \in [n]})$, $\mathbf{X}^\ell = \{x_{i,j}^\ell\}_{i \in [n], j \in [q]}$

$b \leftarrow \{0, 1\}$ $\text{CT}_{i,j} \leftarrow \text{MIFE.Enc}(\text{mpk}, \text{ek}, \mathbf{X})$

Fig. 2. MIFE security game.

The game output β depends on the following conditions. If QKeyGen queries have been issued for some function f and there exists a pair of values (x_0, x_1) queried to QEnc , such that $f(x_0) \neq f(x_1)$, we set $\beta \leftarrow \{0, 1\}$. In any aother case we set the output to $\beta \leftarrow b'$.

\mathcal{A} wins in the game if $\beta = b$ and we remark that a naive adversary, by sampling randomly β has probability of winning equal to $\frac{1}{2}$. We denote the advantage that \mathcal{A} has of winning as $\text{Adv}^{\text{IND}}(\mathcal{A})$ and we say this FE is IND-secure if for any PPT adversary \mathcal{A} , $\text{Adv}^{\text{IND}}(\mathcal{A}) = |\text{Pr}[\beta = 1|b = 1] - \text{Pr}[\beta = 1|b = 0]| \leq \text{negl}(\lambda)$.

The game above captures the adaptive security of FE. There exists also a selective variant, where the adversary is forced to send all its encryption queries QEnc in one shot, before issuing any other type of query.

3.2 Multi Input Functional Encryption

We alter the original definition of MIFE scheme in [24] for consistency purposes. Specifically, we let $\text{MIFE.Setup}(\cdot)$ output additionally a master public key mpk and we augment all remaining algorithms' inputs with mpk .

Definition 5 (Multi-Input Functional Encryption). Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of sets \mathcal{F}_λ of functions $f : \mathcal{X}_\lambda \rightarrow \mathcal{Y}_\lambda$. A multi-input functional encryption scheme (MIFE) for the functionality class \mathcal{F}_λ is a tuple of four algorithms $\text{MIFE} = (\text{MIFE.Setup}, \text{MIFE.KeyGen}, \text{MIFE.Enc}, \text{MIFE.Dec})$.

$\text{MIFE.Setup}(1^\lambda, n)$: Takes as input a unary representation of the security parameter λ and an integer n , and outputs a master public key mpk , a master secret key msk , and a set of n encryption keys $\{\text{ek}_i\}_{i \in [n]}$.

$\text{MIFE.KeyGen}(\text{mpk}, \text{msk}, f)$: Takes as input a master public key mpk , a master secret key msk and a function $f \in \mathcal{F}_\lambda$, and outputs a functional secret key sk_f .

$\text{MIFE.Enc}(\text{mpk}, \text{ek}_i, x)$: Takes as input a master public key mpk , an encryption key ek_i and a string $x \in \mathcal{X}_\lambda$, and outputs a ciphertext ct_i or err (denoting an encryption error).

$\text{MIFE.Dec}(\text{mpk}, f, \text{sk}_f, \{\text{ct}_i\}_{i \in [n]})$: Takes as input a master public key pk_f , a function f , a functional key sk_f , and a set of ciphertexts $\{\text{ct}_i\}_{i \in [n]}$ and outputs a function value $y \in \mathcal{Y}_\lambda$ or \perp , which indicates an invalid ciphertext.

Security of MIFE schemes was defined in [24], parameterized by the number of encryption keys known to the adversary, and the number of challenge messages per encryption key. Similarly to [26], our construction is uplifted of any conditions regarding these two parameters.

Definition 6 (Indistinguishability-based security [26]). We say that a MIFE scheme for n -ary functions \mathcal{F} is fully IND-secure if for every adversary \mathcal{A} , the advantage of \mathcal{A} defined as: $Adv^{sec-MIFE}(\mathcal{A}(1^\lambda)) = |\Pr[IND_{\mathcal{A}}^{MIFE}] - 1/2| < \text{negl}(\lambda)$, where the game $IND_{\mathcal{A}}^{MIFE}(1^\lambda)$ is depicted in Fig. 2.

4 Publicly Auditable Functional Encryption

In this section, we present our definition of publicly auditable functional encryption, its security game, and different flavors of public audibility, each one corresponding to a different corruption set among the encryptor, authority, and decryptor entities.

Below we show the algorithms of a PAFE scheme. The main differences compared to FE are: (i) the addition of a functional public key pk_f for each function, (ii) a new algorithm *ProveDec* executed by the decryptor to convince a third party (auditor) about the decryption correctness, and (iii) a new algorithm *PAFE.VerifyDec* that takes pk_f (but not sk_f) and a proof of decryption correctness for the same function f as the key and accepts or rejects an output.

Definition 7 (Publicly Auditable FE) Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of sets \mathcal{F}_λ of functions $f: \mathcal{X}_\lambda \rightarrow \mathcal{Y}_\lambda$. A publicly auditable functional encryption scheme for the functionality class \mathcal{F}_λ is a tuple of six algorithms $\text{PAFE} = (\text{PAFE.Setup}, \text{PAFE.KeyGen}, \text{PAFE.Enc}, \text{PAFE.Dec}, \text{PAFE.ProveDec}, \text{PAFE.VerifyDec})$.

- $\text{PAFE.Setup}(1^\lambda)$: Takes as input a unary representation of the security parameter λ and outputs a master public key mpk , a master secret key msk , and an encryption key ek .
- $\text{PAFE.KeyGen}(\text{mpk}, \text{msk}, f)$: Takes as input a master public key mpk , a master secret key msk and a function $f \in \mathcal{F}_\lambda$, and outputs a functional secret key sk_f and a functional public key pk_f .
- $\text{PAFE.Enc}(\text{mpk}, \text{ek}, x)$: Takes as input an encryption key ek and a string $x \in \mathcal{X}_\lambda$, and outputs a ciphertext ct .
- $\text{PAFE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct})$: Takes as input a master public key mpk , a function $f \in \mathcal{F}_\lambda$, a functional key sk_f , and a ciphertext ct . It outputs a function value $y \in \mathcal{Y}_\lambda$ or \perp , which indicates an invalid ciphertext.
- $\text{PAFE.ProveDec}(\text{mpk}, f, \text{sk}_f, \text{pk}_f, \text{ct}, y)$: Takes as input a master public key mpk , a function $f \in \mathcal{F}_\lambda$, a functional secret key sk_f , a functional public key pk_f , a ciphertext ct , and a value y and outputs a proof π_d .
- $\text{PAFE.VerifyDec}(\text{mpk}, f, \text{pk}_f, \text{ct}, y, \pi)$: Takes as input a master public key mpk , a function $f \in \mathcal{F}_\lambda$, a functional public key pk_f , a ciphertext ct , a value y , and a proof π and outputs 1 if $y = \text{PAFE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct})$, and 0 otherwise.

PAFE satisfies the correctness of FE schemes as per Definition 3 and additionally satisfies the following notion of auditable correctness.

Auditable correctness. A scheme PAFE has auditable correctness, if $\forall \lambda \in \mathbb{N}$, $\forall f \in \mathcal{F}_\lambda, \exists x \in \mathcal{X}_\lambda$ the following probability is negligible in λ .

$$\Pr \left[\begin{array}{l} \text{PAFE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct}) = f(x) \\ 1 \leftarrow \text{PAFE.VerifyDec}(\text{mpk}, f, \\ \text{pk}_f, \text{ct}, f(x), \pi_d) \end{array} \middle| \begin{array}{l} (\text{msk}, \text{mpk}, \text{ek}) \leftarrow \text{PAFE.Setup}(1^\lambda) \\ \text{ct} \leftarrow \text{PAFE.Enc}(\text{ek}, x) \\ (\text{sk}_f, \text{pk}_f) \leftarrow \text{KeyGen}(\text{mpk}, \text{msk}, f) \\ \pi_d \leftarrow \text{PAFE.ProveDec}(\text{mpk}, f, \text{sk}_f, \text{pk}_f, \\ \text{ct}, f(x)) \end{array} \right]$$

The security of a PAFE scheme is captured by an indistinguishability game that is an “extended” version of the one from Definition 4 to allow also for oracle access to functional public keys and decryptions.

Definition 8 (Security for PAFE). Consider the following game between PPT adversary \mathcal{A} and challenger \mathcal{C} :

- *Initialization:* \mathcal{C} runs the setup algorithm $(\text{mpk}, \text{msk}, \text{ek}) \leftarrow \text{PAFE.Setup}(1^\lambda)$ and chooses a random bit $b \leftarrow_s \{0, 1\}$. It provides the master public key mpk to \mathcal{A} .
- *Encryption queries:* $\text{QEnc}(x_0, x_1)$: \mathcal{A} has unlimited adaptive access to a Left-or-Right encryption oracle and receives ciphertext $\text{ct}^b \leftarrow \text{PAFE.Enc}(\text{mpk}, \text{ek}, x_b)$.
- *Functional key queries* $\text{QSKeyGen}(f)$: \mathcal{A} has unlimited and adaptive oracle access to the $\text{PAFE.KeyGen}(\text{mpk}, \text{msk}, f)$ algorithm for input functions f of its choice. It receives the functional secret and public keys $(\text{sk}_f, \text{pk}_f)$.
- *Functional public key queries* $\text{QPKeyGen}(f)$: \mathcal{A} has unlimited and adaptive oracle access to the $\text{PAFE.KeyGen}(\text{mpk}, \text{msk}, f)$ algorithm for any input function f of its choice. It is given back a functional public key pk_f .
- *Decryption queries* $\text{QDec}(\text{ct}^b, f)$: \mathcal{A} has unlimited adaptive access to an oracle for the $\text{PAFE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct})$ algorithm, for any input ciphertext ct and any function f of its choice. If no $\text{Q}(S/P)\text{KeyGen}$ query for f has been issued, \mathcal{C} outputs \perp . Otherwise, \mathcal{A} receives the function output $f(x_b)$, where $\text{ct}_b = \text{PAFE.Enc}(\text{mpk}, \text{ek}, x_b)$, alongside a proof π about its correctness.
- *Finalize:* \mathcal{A} provides its guess b' on the bit b and this procedure outputs the result β of the security game, according to the analysis given below.

\mathcal{A} wins in the game if $\beta = b$ and we remark that a naive adversary, by sampling randomly β has probability of winning equal to $\frac{1}{2}$. In case \mathcal{A} has either issued (i) QSKeyGen queries for some function f and also QEnc queries for a pair of values, (x_0, x_1) , such that $f(x_0) \neq f(x_1)$, or (ii) QDec query for ciphertext ct_b and function f and has issued a $\text{ct}^b \leftarrow \text{QEnc}(x_0, x_1)$ query, where $\text{ct}_0 = \text{PAFE.Enc}(\text{mpk}, \text{ek}, x_0)$ and $\text{ct}_1 = \text{PAFE.Enc}(\text{mpk}, \text{ek}, x_1)$, such that $f(x_0) \neq f(x_1)$, then $\beta \leftarrow_s \{0, 1\}$. Otherwise $\beta = b'$. We denote the advantage that \mathcal{A} has of winning as $\text{Adv}^{\text{sec-PAFE}}(\mathcal{A})$ and we say this PAFE is secure if for any PPT adversary \mathcal{A} , $\text{Adv}^{\text{sec-PAFE}}(\mathcal{A}) = |\text{Pr}[\beta = 1|b = 1] - \text{Pr}[\beta = 1|b = 0]| \leq \text{negl}$.

As with FE, we can also have a selective version of the above game where the adversary declares all its encryption queries prior to other oracle accesses.

<p>PA-UD($1^\lambda, \mathcal{A}$) and PA-UAD($1^\lambda, \mathcal{A}$)</p> <p>$(\text{mpk}, \text{msk}, \text{ek}) \leftarrow \text{PAFE.Setup}(1^\lambda)$</p> <p>$(\text{mpk}^*, \text{ek}^*) \leftarrow \mathcal{A}(1^\lambda)$</p> <p>$(y^*, \pi^*, i, j) \leftarrow \mathcal{A}^{\text{PAFE.KeyGen}(\text{mpk}, \text{msk}, \cdot), \text{PAFE.Enc}(\text{mpk}, \text{ek}, \cdot)}(1^\lambda, \text{mpk})$</p> <p>▷ i corresponds to the i-th PAFE.KeyGen oracle query and $(f_i, \text{sk}_{f_i}, \text{pk}_{f_i})$ is its associated information</p> <p>▷ j corresponds to the j-th PAFE.Enc oracle query and (x_j, ct_j) is its associated information</p> <p>If $(y^* \neq f_i(x_j) \wedge \text{PAFE.VerifyDec}(\text{mpk}, f_i, \text{pk}_{f_i}, \text{ct}_j, y^*, \pi^*) = 1)$</p> <p>If $(y^* \neq f^*(x_j) \wedge \text{PAFE.VerifyDec}(\text{mpk}^*, f^*, \text{pk}_{f^*}^*, \text{ct}_j, y^*, \pi^*) = 1)$</p> <p>Output 1</p> <p>Else output 0</p>
--

Fig. 3. Public Auditability with Untrusted (Authority and) Decryptor (Color figure online)

<p>PA-UED($1^\lambda, \mathcal{A}$) and PA-UEAD($1^\lambda, \mathcal{A}$)</p> <p>$(\text{mpk}, \text{msk}, \text{ek}) \leftarrow \text{PAFE.Setup}(1^\lambda)$</p> <p>$(\text{mpk}^*, \text{ct}^*, \{y_i^*, \pi_i^*, f_i^*, \text{pk}_{f_i}^*\}_{i \in [n]}) \leftarrow \mathcal{A}(1^\lambda)$</p> <p>$(y^*, \pi^*, \text{ct}^*, i) \leftarrow \mathcal{A}^{\text{PAFE.KeyGen}(\text{mpk}, \text{msk}, \cdot)}(1^\lambda, \text{mpk}, \text{ek})$</p> <p>▷ i corresponds to the i-th PAFE.KeyGen oracle query and $\{f_i, \text{sk}_{f_i}, \text{pk}_{f_i}\}$ is its associated information</p> <p>If $(\nexists x \in \mathcal{X}_\lambda : y^* = f_i(x) \wedge \text{PAFE.VerifyDec}(\text{mpk}, f_i, \text{pk}_{f_i}, \text{ct}^*, y^*) = 1)$</p> <p>If $(\nexists x' \in \mathcal{X}_\lambda : \forall i \in [n] y_i^* = f_i(x') \wedge \text{VerifyDec}(\text{mpk}^*, \text{ct}^*, f_i^*, \text{pk}_{f_i}^*, y_i^*, \pi_i^*) = 1)$</p> <p>Output 1</p> <p>Else output 0</p>

Fig. 4. Public Auditability with Untrusted Encryptor, (Authority, and) Decryptor. (Color figure online)

4.1 Public Auditability Definitions

There exist totally four different cases of public auditability, depending on which of the involved parties are adversarial and possibly colluding to trick the auditor into accepting an incorrect functional decryption result (see Table 1). In particular we consider the cases where: (i) only the decryptor is corrupted (PA-UD), (ii) both the encryptor and the decryptor are corrupted (PA-UED), (iii) both the key-generating authority and the decryptor are corrupted (PA-UAD), and (iv) the encryptor, the key-generating authority, and the decryptor are corrupted (PA-UEAD). Each corruption set results in a different definition of public auditability, defined below. At a high level, auditability ensures that, even though auditors have no decryption capabilities of their own, there is still a guarantee of (*at least* some level of) consistency between what they receive and what the decryptor computed. We note that this renders pointless to define this property for cases where the decryptor is honest (or the auditor itself is dishonest).

We provide game-based definition of public auditability for all four corruption cases. The two games for honest (resp. corrupted) authority are very similar, hence we depict them together in Fig. 3 (resp. Figure 4). The lines shown in red

correspond to the games modified for corrupted authorities, each time replacing the preceding line in black and we denote by the superscript \star all adversary-provided elements.

Definition 9. (Public Auditability) *Let $\text{SET} \in \{\text{UD}, \text{UAD}, \text{UED}, \text{UEAD}\}$ and consider the experiments PA-UD, PA-UAD from Figure 3 and PA-UED, PA-UEAD from Figure 4. A PAFE for \mathcal{F} is SET-publicly auditable against the corresponding set of corruptions if $\forall \lambda \in \mathbb{N}, \forall x \in \mathcal{X}_\lambda, \forall \text{PPT adversaries } \mathcal{A}$, it holds that $\Pr[\text{PA-SET}(1^\lambda, \mathcal{A}) = 1] < \text{negl}(\lambda)$.*

Based on the above definition we make the following observations. First, when the encryptor is honest, the auditor should be guaranteed to receive the exact function output corresponding to the function f_i of its choice and the actual plaintext x used by the honest encryptor. On the contrary, for cases PA-UED and PA-UEAD where the encryptor is not trusted, there is no guarantee that the ciphertexts were honestly computed. In these settings, the best the auditor can hope for is a notion of “consistency.” I.e., there must exist some common plaintext element within the domain of f_i (or, in the case of multiple functions, the non-empty intersection of their respective domains) that evaluates to the output (or outputs) received and verified by the auditor.

Second, in the PA-UD experiment the condition $y^\star \neq f_i(x_j)$ could be equivalently written as $(y^\star \neq \text{PAFE.Dec}(\text{mpk}, f_i, \text{sk}_{f,i}, \text{ct}_j))$. However, this is not the case for PA-UAD. In the former case the encryptor chooses an $x \in \mathcal{X}_\lambda$ and encrypts honestly. In the latter case the encryptor may still intend to encrypt a valid plaintext x . However, the adversary (colluding authority and decryptor) may manipulate the generated keys so that even an honestly generated ciphertext is not decryptable into the image space of f_i .

5 PAFE Constructions from FE

Below we provide constructions for secure and publicly auditable PAFE schemes from secure FE schemes and other cryptographic primitives i.e., commitment schemes and NIWI proof systems. As in Sect. 4, we “group” our constructions together. However, now we do so based on whether the key-generating authority is trusted or not. Lines in blue in all figures of this section, depict *additional* steps/parts for when the encryptor is untrusted. Both auditable and regular correctness, for all our constructions, follow directly from the correctness of the underlying FE scheme, the correctness of the commitment scheme and the completeness of the employed NIWI proof systems. For each construction we provide proofs about its security and public auditability flavor. For readability reasons our proofs are delegated to the Appendix and our full version [4].

5.1 Auditability with Trusted Authority

PA-UD Auditability. For this construction (Fig. 5) we employ an FE scheme, a perfectly binding and computationally hiding commitment scheme, and a perfectly sound and computationally witness-indistinguishable NIWI proof system.

<p>PAFE.Setup(1^λ):</p> $s_1, s_2, u_d, u_e \leftarrow \mathcal{S} \{0, 1\}^\lambda$ $pp \leftarrow \text{Com.Setup}(1^\lambda; s_1)$ $(\text{msk}, \text{mpk}, \text{ek}) \leftarrow \text{FE.Setup}(1^\lambda; s_2)$ $c_d \leftarrow \text{Com.Commit}(\text{msk}; u_d)$ $c_e \leftarrow \text{Com.Commit}(\text{msk}; u_e)$ Return $(pp, \text{mpk}, \text{msk}, \text{ek}, c_d, c_e)$ <p>PAFE.KeyGen($\text{mpk}, \text{msk}, f$):</p> $\text{sk}_f \leftarrow \text{FE.KeyGen}(\text{mpk}, \text{msk}, f)$ $r_f \leftarrow \mathcal{S} \{0, 1\}^\lambda$ $\text{pk}_f \leftarrow \text{Com.Commit}(\text{sk}_f; r_f)$ Return $(\text{sk}_f, r_f, \text{pk}_f)$	<p>PAFE.Enc($\text{mpk}, \text{msk}, \text{ek}, x, c_e, u_e$):</p> $\text{ct} \leftarrow \text{FE.Enc}(\text{ek}, x; s_e), s_e \leftarrow \mathcal{S} \{0, 1\}^\lambda$ $\pi_e \leftarrow \text{NIWI}_e.\text{Prove}(\text{mpk}, \text{msk}, \text{ek}, x, \text{ct}, c_e, u_e, s_e)$ Return (ct, π_e) <p>PAFE.Dec($\text{mpk}, f, \text{sk}_f, \text{ct}$):</p> $y \leftarrow \text{FE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct})$ Return y <p>$\pi_d \leftarrow \text{NIWI}_d.\text{Prove}(\text{mpk}, \text{msk}, f, \text{sk}_f, r_f, \text{pk}_f, \text{ct}, y, c_d, u_d)$ Return π_d</p> <p>PAFE.VerifyDec($\text{mpk}, f, \text{pk}_f, \text{ct}, y, c_e, c_d, \pi_e, \pi_d$):</p> $b_1 \leftarrow \text{NIWI}_d.\text{Verify}(\text{mpk}, f, \text{pk}_f, \text{ct}, y, c_d, \pi_d)$ $b_2 \leftarrow \text{NIWI}_e.\text{Verify}(\text{mpk}, \text{ct}, c_e, \pi_e)$ Return $b_1 \wedge b_2$
---	--

Fig. 5. PAFE construction with untrusted (encryptor and) decryptor. (Color figure online)

<p>Relation $R_{UD,d}$:</p> Statement: $z_d = (\text{mpk}, f, \text{pk}_f, \text{ct}, y, c_d)$ Witness: $w_d = (\text{sk}_f, r_f, \top, u_d)$ $R_{UD,d}(z_d, w_d) = 1$ iff: $(\text{pk}_f \leftarrow \text{Com.Commit}(\text{sk}_f; r_f)$ $\wedge y \leftarrow \text{FE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct}))$ $\vee c_d \leftarrow \text{Com.Commit}(\top; u_d)$	<p>Relation $R_{UD,e}$:</p> Statement: $z_e = (\text{mpk}, \text{ct}, c_e)$ Witness: $w_e = (\text{ek}, x, s_e, \top, u_e)$ $R_{UD,e}(z_e, w_e) = 1$ iff: $\text{ct} = \text{FE.Enc}(\text{mpk}, \text{ek}, x; s_e)$ $\vee c_e \leftarrow \text{Com.Commit}(\top; u_e)$
---	---

Fig. 6. Relations used in the constructions of Fig. 5. (Color figure online)

First, our functional public keys pk_f are computed as commitments of the corresponding sk_f . Second, PAFE.ProveDec produces a NIWI proof for the correctness of the decryption. The relation R_{UD} is shown in Fig. 6. Symbol \top , included in this relation, is a fixed value from the supported domain of the commitment scheme Com , lying outside the domain of possible msk values that can be produced from FE.Setup (assuming, without loss of generality, that the domain of Com is a superset of the latter). This turns NIWI proofs into “OR”-proofs, allowing us to formally prove the IND-security of our scheme even in the presence of arbitrary oracle queries, without compromising auditability—since PAFE.Setup , which is executed by the trusted authority, will never produce \top as the msk . Below, we state Theorem 1 regarding the security and PA-UD public auditability of our construction, whose full proof we include in Appendix A.

Theorem 1. *Let FE be an IND-secure FE for a family of functions \mathcal{F} . Let Com be a perfectly binding and computationally hiding commitment scheme and NIWI_d a perfectly sound and computationally witness-indistinguishable NIWI for relation R_{UD} (Fig. 6). Then, the PAFE scheme of Fig. 5 for \mathcal{F} is secure as per Definition 8 and UD-publicly auditable as per Definition 9.*

PA-UED Auditability. To achieve public auditability against untrusted encryptor and decryptor we use the our PA-UD construction as a baseline, but additionally we require the encryptor to provide a NIWI proof for the validity of the ciphertext computation. The auditor now has to perform two NIWI verifications while executing the PAFE.VerifyDec algorithm. If both succeed, then it is convinced about the legitimacy of the received result. The additional elements this construction has with respect to the the PA-UD one are highlighted in blue in Fig. 5 and the two relations $R_{UED,e}$ and $R_{UED,d}$ that need to be supported by the NIWI schemes are defined in Fig. 8. Below we state Theorem 2 and we provide its proof in the full version of our work [4].

Theorem 2. *Let FE be an IND-secure FE scheme for a family of functions \mathcal{F} . Let Com be a perfectly binding and computationally hiding commitment scheme and $NIWI_d, NIWI_e$ perfectly sound and computationally witness indistinguishable NIWI proof systems for relations $R_{UED,d}, R_{UED,e}$ (Fig. 6). Then, the PAFE scheme of Fig. 5 for \mathcal{F} is adaptively secure under Definition 8 and publicly auditable against untrusted encryptors and decryptors as per Definition 9.*

5.2 Auditability with Untrusted Authority

Contrary to our previous constructions, since the authority can no longer be trusted to honestly run the setup and key-generation algorithms, our “OR” proof strategy no longer works. Instead, we adapt an approach from [11] based on replicating the FE computations, while accepting the majority of the functional decryptions as the correct output. In particular, our constructions use four instances of an IND-secure FE scheme, again combined with a perfectly binding and computationally hiding commitment scheme, and perfectly sound and computationally witness-indistinguishable NIWIs.

PA-UAD. To achieve public auditability in the presence of untrusted authority and decryptor, we augment the output of the PAFE.KeyGen algorithm to include a NIWI proof that guarantees at least three-out-of-four FE.Setup and FE.KeyGen instances have been generated honestly (Fig. 7). Additionally, the output of the PAFE.ProveDec algorithm now states that at least two-out-of-the-four decryptions have been executed honestly. The corresponding relations $R_{UAD,f}, R_{UAD,d}$ that need to be supported by the NIWI schemes, are defined in Fig. 8. What is more, PAFE.Dec now returns the majority of the functional decryptions—in case majority is not reached, PAFE.Dec returns \perp .

Auditability now is based on the fact that: (i) the encryptor is assumed to be trusted, meaning that it always encrypts the same message (regardless of how the encryption keys were originally generated), (ii) it is always guaranteed that *at least one* of the four FE instances is executed honestly with respect to key-generation, encryption, and decryption. To justify this in more detail, note that since three-out-of-four FE keys are computed correctly and two-out-of-four decryptions are performed correctly (and given the encryptor uses the same

<p>PAFE.Setup(1^λ): For $i \in [4]$: $(\text{msk}_i, \text{mpk}_i, \text{ek}_i) \leftarrow \text{FE.Setup}(1^\lambda; s_i)$ $s_i \leftarrow \{0, 1\}^\lambda$ $\text{c}_f \leftarrow \text{Com.Commit}(1^{4 \cdot \text{len}}; u_f)$, $u_f \leftarrow \{0, 1\}^\lambda$ $\text{c}_e \leftarrow \text{Com.Commit}(1^{\text{len}}; u_e)$ $u_e \leftarrow \{0, 1\}^\lambda$ $\text{c}_d \leftarrow \text{Com.Commit}(1^{\text{len}}; u_d)$ $u_d \leftarrow \{0, 1\}^\lambda$ $\text{msk} = \{\text{msk}_i\}_{i \in [4]}$ $\text{mpk} = (\{\text{mpk}_i\}_{i \in [4]}, \text{c}_f, \text{c}_e, \text{c}_d)$ $\text{ek} = \{\text{ek}_i\}_{i \in [4]}$ Return $(\text{mpk}, \text{msk}, \text{ek})$</p> <p>PAFE.KeyGen($\text{mpk}, \text{msk}, f$): For $i \in [4]$: $\text{sk}_{f,i} \leftarrow \text{FE.KeyGen}(\text{mpk}_i, \text{msk}_i, f; s_{k,i})$ $s_{k,i} \leftarrow \{0, 1\}^\lambda$ $\text{pk}_{f,i} \leftarrow \text{Com.Commit}(\text{sk}_{f,i}; r_{f,i})$ $r_{f,i} \leftarrow \{0, 1\}^\lambda$ $s_k = \{s_{k,i}\}_{i \in [4]}$ $r_f = \{r_{f,i}\}_{i \in [4]}$ $\text{sk}_f = \{(\text{sk}_{f,i}, r_{f,i}, s_{k,i})\}_{i \in [4]}$ $\text{pk}_f = \{\text{pk}_{f,i}\}_{i \in [4]}$ $\pi_f \leftarrow \text{NIWI}_f.\text{Prove}(\text{mpk}, \text{msk}, f, \text{sk}_f, \text{pk}_f, s_k, r_f)$ $\text{pk}'_f = (\text{pk}_f, \pi_f)$ Return $(\text{sk}_f, \text{pk}'_f)$</p>	<p>PAFE.Enc(mpk, ek, x): For $i \in [4]$: $\text{ct}_i \leftarrow \text{FE.Enc}(\text{ek}_i, x; s_{e,i})$ $s_{e,i} \leftarrow \{0, 1\}^\lambda$ $s_e = \{s_{e,i}\}_{i \in [4]}$ $\pi_e \leftarrow \text{NIWI}_e.\text{Prove}(\text{mpk}, \text{ek}, x, \text{ct}, s_e)$ Return $\text{ct} = (\{\text{ct}_i\}_{i \in [4]}, \pi_e)$</p> <p>PAFE.Dec($\text{mpk}, f, \text{sk}_f, \text{ct}$): For $i \in [4]$: $y_i \leftarrow \text{FE.Dec}(\text{mpk}_i, f, \text{sk}_{f,i}, \text{ct}_i)$ If $\exists j_1 \neq j_2 \neq j_3 \in [4]: y_{j_1} = y_{j_2} = y_{j_3}$ Return y_1 Return \perp</p> <p>PAFE.ProveDec($\text{mpk}, f, \text{sk}_f, \text{pk}_f, \text{ct}, y$): $\pi_d \leftarrow \text{NIWI}_d.\text{Prove}(f, y, \text{mpk}, \text{sk}_f, \text{pk}_f, \text{ct})$ Return π_d</p> <p>PAFE.VerifyDec($\text{mpk}, \text{pk}'_f, \text{ct}, \pi_d, y, f$): Parse $\text{pk}'_f = (\text{pk}_f, \pi_f)$ $b_1 \leftarrow \text{NIWI}_f.\text{Verify}(\text{mpk}, \text{pk}'_f, f)$ $b_2 \leftarrow \text{NIWI}_e.\text{Verify}(\text{mpk}, \text{ct})$ $b_3 \leftarrow \text{NIWI}_d.\text{Verify}(\text{mpk}, \text{pk}_f, \text{ct}, y, \pi_d, f)$ Return $b_1 \wedge b_2 \wedge b_3$</p>
--	---

Fig. 7. PAFE construction with untrusted [encryptor](#), authority, and decryptor.

plaintext for all four ciphertexts), there is no way the sets of FE instances with correct keys and those with correct decryptions are disjoint.

More technically, this translates into the following: Suppose the decryptor provides the auditor with a fabricated y^* that *does not* correspond to the output of the $\text{FE.Dec}(\cdot)$ for any of the correctly computed $(\text{mpk}, \text{sk}_f)$. This would result in the decryptor breaking the correctness of the underlying FE scheme as all ciphertexts are also correctly computed. Hence, assuming the FE scheme is perfectly correct the only output y for which the auditor who runs PAFE.VerifyDec will accept is the correct functional output for the x encrypted by the encryptor. We now state the following theorem and we offload the formal proof to the full version of our work [4].

Theorem 3. *Let FE be a secure FE scheme for a family of functions \mathcal{F} . Let Com be a perfectly binding and computationally hiding commitment scheme, NIWI_f and NIWI_d perfectly sound and computationally witness-indistinguishable NIWIs for $R_{\text{UAD},d}$ and $R_{\text{UAD},d}$ respectively (Fig. 8). Then, PAFE in Fig. 7 is adaptively secure under Definition 8 and publicly auditable against untrusted authority and decryptor as per Definition 9.*

PA-UEAD. Finally we focus on the “extreme” case where all protocol participants (authority, encryptor, and decryptor) collaborate to cheat the auditor into accepting an incorrect functional output. To achieve public auditability in this case, we use the PA-UAD construction as a baseline but we augment the output of the encryption to include a NIWI proof as well, following a similar approach as the one we adopted to go from PA-UD to PA-UED. However, given the multiple FE instances we need to further modify our technique (Fig. 7).

Overall, we use NIWIs to prove that at least three-out-of-the-four executions of the FE.Setup and FE.KeyGen algorithms have been executed honestly. Likewise, for at least two-out-of-the-four executions of the FE.Enc and at least three-out-of-the-four executions of FE.Dec algorithms. The above are depicted in relations $R_{UEAD,f}$, $R_{UEAD,e}$ and $R_{UEAD,d}$ in Fig. 8. Unlike in the PA-UAD case, here we additionally consider the encryptor to be untrusted therefore we cannot use the same reasoning to guarantee that at least for one FE instance all the algorithms (FE.Setup, FE.KeyGen, FE.Enc, FE.Dec), have been correctly executed. Instead, we reason about the properties of our construction as follows:

First, $R_{UEAD,f}$ and $R_{UEAD,d}$ both require that pk_f is computed honestly, for their respective indices, which “ties them together”. On the other hand, the encryption algorithm is not tied to the pk_f . However, having a three-out-of-four threshold for the NIWI proof regarding functions key generation suffices to guarantee majority in PAFE.Dec. Finally, note that $R_{UEAD,f}$ enforces that all four decryptions return the same result which guarantees that the auditor receives a valid function output. Recall that, since the encryptor is not assumed honest in this setting, we cannot guarantee that the output y is the correct functional output for *the* plaintext x used by for all four ciphertexts; just a “consistent” explanation for all four of them (see Sect. 4.1).

One downside of our final construction is that it only satisfies the selective version of Definition 8, where the adversary issues all encryption queries before any other oracle access (but after receiving the output of PAFE.Setup). To see why this is necessary, observe that condition (2) of relation $R_{UEAD,f}$ requires knowledge of all ciphertexts ct_i for the NIWI proof generation. Similarly to before, we state the following theorem and delegate the proof for our PA-UEAD scheme to the full version of our work [4].

Theorem 4. *Let FE be an IND-secure FE scheme for a family of functions \mathcal{F} . Let Com be a perfectly binding and computationally hiding commitment scheme, NIWI_f, NIWI_e, NIWI_d computationally witness-indistinguishable and perfectly sound NIWIs for relations $R_{UEAD,e}$, $R_{UEAD,f}$, and $R_{UEAD,d}$ shown in Fig. 8. Then, PAFE in Fig. 7 is selectively secure under Definition 8 and publicly auditable against untrusted encryptor, authority, and decryptor as per Definition 9.*

<p>Relation $R_{UAD,f}$: Statement: $z_f = (\{\text{mpk}_i, f_i, \text{pk}_{f,i}, \text{ct}_i\}_{i \in [4]});$ Witness: $w_f = (\{\text{msk}_i, \text{sk}_{f,i}, r_{f,i}, r_{k_i}, s_i\}_{i \in [4]})$ $R_{UAD,f}(z_f, w_f) = 1$ iff either of the following conditions hold: (1) - $\forall j \in [4] : \text{pk}_{f,j} \leftarrow \text{Com}(\text{sk}_{f,j}; r_{f,j}) \wedge (\text{mpk}_j, \text{msk}_j) \leftarrow \text{FE.Setup}(1^\lambda; s_j)$ $\wedge \text{sk}_{f,j} \leftarrow \text{FE.KeyGen}(\text{mpk}_j, \text{msk}_j, f_j; r_{k_j})$ (2) - $\exists A_1 \subset [4],$ with $A_1 = 3,$ s.t. $\forall j \in A_1: \text{pk}_{f,j} \leftarrow \text{Com.Commit}(\text{sk}_{f,j}; r_{f,j})$ $\wedge (\text{mpk}_j, \text{msk}_j) \leftarrow \text{FE.Setup}(1^\lambda; s_j) \wedge \text{sk}_{f,j} \leftarrow \text{FE.KeyGen}(\text{mpk}_j, \text{msk}_j, f; r_{k_j})$</p> <p>Relation $R_{UAD,d}$: Statement: $z_d = (\{\text{mpk}_i, f_i, \text{pk}_{f,i}, \text{ct}_i\}_{i \in [4]}, y, \pi_d)$ Witness: $w_d = (\{\text{sk}_{f,i}, r_{f,i}\}_{i \in [4]})$ $R_{UAD,d}(z_d, w_d) = 1$ iff either of the following conditions hold: (1) $\forall k \in [4] : \text{pk}_{f_k} \leftarrow \text{Com}(\text{sk}_{f_k}; r_{f_k}) \wedge y \leftarrow \text{FE.Dec}(\text{mpk}_k, f, \text{sk}_{f_k}, \text{ct}_k)$ (2) $\exists A_2 \subset [4],$ with $A_2 = 2,$ s.t. $\forall k \in A_2: \text{pk}_{f_k} \leftarrow \text{Com}(\text{sk}_{f_k}; r_{f_k})$ $\wedge y \leftarrow \text{FE.Dec}(\text{mpk}_k, f, \text{sk}_{f_k}, \text{ct}_k) \wedge c_f \leftarrow \text{Com}(\{\text{sk}_{f,i}\}_{i \in [4]}; u_f) \wedge c_d \leftarrow \text{Com}(0^{\text{len}}; u_d)$</p> <p>Relation $R_{UEAD,f}$: Statement: $z_f = \{\text{mpk}_i, f_i, \text{pk}_{f,i}, \text{ct}_i\}_{i \in [4]}, c_d, c_f);$ Witness: $w_f = (\{\text{msk}_i, \text{sk}_{f,i}, r_{f,i}, s_i\}_{i \in [4]}, y, u_d, u_f)$ $R_{UEAD,f}(z_f, w_f) = 1$ iff either of the following conditions hold: (1) - $(\forall j \in [4] : \text{pk}_{f,j} \leftarrow \text{Com.Commit}(\text{sk}_{f,j}; r_{f,j}) \wedge (\text{mpk}_j, \text{msk}_j) \leftarrow \text{FE.Setup}(1^\lambda; s_j)$ $\wedge \text{sk}_{f,j} \leftarrow \text{FE.KeyGen}(\text{mpk}_j, \text{msk}_j, f_j; r_{f,j})) \wedge c_f \leftarrow \text{Com.Commit}(1^{4 \cdot \text{len}}; u_f)$ (2) - $(\exists A_1 \subset [4],$ with $A_1 = 3,$ s.t. $\forall j \in A_1:$ $\text{pk}_{f,j} \leftarrow \text{Com.Commit}(\text{sk}_{f,j}; r_{f,j}) \wedge (\text{mpk}_j, \text{msk}_j) \leftarrow \text{FE.Setup}(1^\lambda; s_j)$ $\wedge \text{sk}_{f,j} \leftarrow \text{FE.KeyGen}(\text{mpk}_j, \text{msk}_j, f; r_{f,j})) \wedge c_f \leftarrow \text{Com.Commit}(0^{4 \cdot \lambda}; u_f)$ $\wedge \exists y \in \mathcal{X}_\lambda$ such that $\forall i \in [4]: y \leftarrow \text{FE.Dec}(\text{mpk}_i, f_i, \text{sk}_{f,i}, \text{ct}_i)$</p> <p>Relation $R_{UEAD,e}$: Statement: $z_e = (\{\text{mpk}_i, \text{ct}_i\}_{i \in [4]}, y, c_e);$ Witness: $w_e = (\{\text{ek}_i, x_i, s_{e,i}\}_{i \in [4]}, u_e)$ $R_{UEAD,e}(z_e, w_e) = 1$ iff either of the following conditions hold: (1) $\forall k \in [4] :$ $\text{ct}_j = \text{FE.Enc}(\{\text{ek}_j, \text{mpk}_j\}, x; s_{e,j}) \wedge c_f \leftarrow \text{Com.Commit}(1^{4 \cdot \text{len}}; u_f)$ $\wedge c_e \leftarrow \text{Com.Commit}(1^{\text{len}}; u_e)$ (2) $\exists A_2 \subset [4],$ with $A_2 = 2,$ s.t. $\forall k \in A_2:$ $\text{ct}_k = \text{FE.Enc}(\{\text{ek}_k, \text{mpk}_k\}, x; s_{e,k}) \wedge c_e \leftarrow \text{Com.Commit}(0^{\text{len}}; u_e)$</p> <p>Relation $R_{UEAD,d}$: Statement: $z_d = (\{\text{mpk}_i, f_i, \text{pk}_{f,i}, \text{ct}_i\}_{i \in [4]}, y, \pi_d);$ Witness: $w_d = (\{\text{sk}_{f,i}, r_{f,i}\}_{i \in [4]})$ $R_{UEAD,d}(z_d, w_d) = 1$ iff either of the following conditions hold: (1) $\forall k \in [4] :$ $\text{pk}_{f_k} \leftarrow \text{Com.Commit}(\text{sk}_{f_k}; r_{f_k}) \wedge y \leftarrow \text{FE.Dec}(\text{mpk}_k, f_k, \text{sk}_{f_k}, \text{ct}_k)$ $\wedge c_e \leftarrow \text{Com.Commit}(1^{\text{len}}; u_f)$ (2) $\exists A_3 \subset [4],$ with $A_3 = 3,$ s.t. $\forall k \in A_3:$ $\text{pk}_{f_k} \leftarrow \text{Com.Commit}(\text{sk}_{f_k}; r_{f_k}) \wedge y \leftarrow \text{FE.Dec}(\text{mpk}_k, f_k, \text{sk}_{f_k}, \text{ct}_k)$ $\wedge c_d \leftarrow \text{Com.Commit}(0^{\text{len}}; u_d)$</p>

Fig. 8. Relations used in the constructions for secure PA-UAD and PA-UEAD PAFE.

6 Relations Among PA Definitions

Here, we investigate any implications between the public auditability definitions. The strongest out of the four is the one where all entities are untrusted, whereas the weakest one is the one where just the decryptor is untrusted. A natural

assumption would be that the strongest implies any weaker definition. However, this is not the case and below we elaborate more on the reason why.

PA-UAD \implies PA-UD: In order to prove this we consider the contrapositive, that is assuming the existence of an adversary \mathcal{A} that wins in the PA-UD with non-negligible probability, we will construct an adversary \mathcal{A}' that wins in the PA-UAD with also non-negligible probability. \mathcal{A} , additionally to the needs of \mathcal{A}' requires an honestly generated mpk and oracle access to the key-generating algorithm. \mathcal{A}' runs `PAFE.Setup` honestly once and provides mpk to \mathcal{A} , and whenever the latter issues a `QKeyGen` query \mathcal{A}' simulates the random oracle. It runs `PAFE.KeyGen` honestly, forwards the output to \mathcal{A} , and stores the input-output information so it can answer future repeated queries. Clearly both \mathcal{A} and \mathcal{A}' have equal advantage of winning their respective games, which concludes our analysis.

PA-UEAD \implies PA-UED: Similarly, to prove this we consider the contrapositive, that is assuming the existence of an adversary \mathcal{A} who wins in the PA-UED with non-negligible probability, we construct an adversary \mathcal{A}'' that wins in the PA-UEAD with also non-negligible probability. The proof is similar to the previous reduction as \mathcal{A}'' operates exactly as \mathcal{A}' . Therefore, both \mathcal{A} and \mathcal{A}'' have equal advantage of winning their respective games.

For the remaining relations, namely **PA-UED \implies PA-UD** and **PA-UEAD \implies PA-UAD** we observe that the winning conditions for adversaries on either side of the implications are different. Specifically, when the encryptor is trusted it produces a legitimate ciphertext that is decryptable to a functional output. Thus, public auditability is satisfied only if the auditor receives that exact function output. However, when the encryptor is untrusted, it suffices for the ciphertext to be decryptable to any function output. This discrepancy in the winning conditions obscures the remaining relations significantly.

7 Publicly Auditable Inner-Product MIFE

Here we present a publicly auditable MIFE scheme for inner-product functionalities, i.e., for weighted sums between plaintext vectors $\mathbf{x} = \{x_i\}_{i \in [n]} \in \mathbb{Z}_q^n$ and weights $\{w_i\}_{i \in [n]} \in \mathbb{Z}_q^n$. We note here that the main difference between FE and MIFE is that the latter allows computing a function over multiple (n) ciphertexts, all of which are individually computed with different encryption keys.

To realise our construction, we employ the existing inner-product multi-input scheme of [8]. At a high level, that construction produces “ElGamal-style” ciphertexts, as follows. Given a vector of inputs $\mathbf{x} = \{x_i\}_{i \in [n]} \in \mathbb{Z}_q^n$, the corresponding ciphertexts are of the form $(g^r, h^r, \{g^{x_i} \cdot h_i^r\}_{i \in [n]})$, for encryption key $\text{ek} = (\{s_i, t_i\}_{i \in [n]})$, and master public key $\text{mpk} = (\mathbb{G}, g, h, \{h_i\}_{i \in [n]})$. The decryptor multiplies all ciphertexts and divides the product by $(g^r)^{\text{sk}_{f,g}} \cdot (h^r)^{\text{sk}_{f,h}}$, where $\text{sk}_{f,g} = \sum_{i=1}^n s_i \cdot w_i$ and $\text{sk}_{f,h} = \sum_{i=1}^n t_i \cdot w_i$, and then solves the discrete logarithm problem to acquire the inner product output (assuming a “small” domain for x_i and n , so that the final discrete logarithm can be computed efficiently [8]).

<p>PAFE.Setup($1^\lambda, 1^n$):</p> <p>Choose cyclic group \mathbb{G} of prime order $q > 2^\lambda$ with generators $g, h \leftarrow \mathbb{G}$</p> <p>For $i \in [n]$:</p> <p>$s_i, t_i \leftarrow \mathbb{Z}_p$</p> <p>$h_i = g^{s_i} \cdot h^{t_i}$</p> <p>$ek = \{(s_i, t_i)\}_{i \in [n]}$</p> <p>$msk = \{(s_i, t_i)\}_{i \in [n]}$</p> <p>$c_d = g^{\sum_{i=0}^n s_i \cdot t_i} \cdot h^{\sum_{i=0}^n s_i \cdot t_i}$</p> <p>$mpk = (\mathbb{G}, g, h, c_d, \{h_i\}_{i \in [n]})$</p> <hr/> <p>PAFE.KeyGen($msk', mpk', \{w_i\}_{i \in [n]}$):</p> <p>Parse $msk' = \{(s_i, t_i)\}_{i \in [n]}$</p> <p>Parse $mpk' = (\mathbb{G}, g, h, c_d, \{h_i\}_{i \in [n]})$</p> <p>$sk_f = (g^{\sum_{i=1}^n s_i \cdot w_i}, h^{\sum_{i=1}^n t_i \cdot w_i})$</p> <p>$pk_f = (g^{\sum_{i=1}^n s_i \cdot w_i}, h^{\sum_{i=1}^n t_i \cdot w_i})$</p> <hr/> <p>PAFE.Enc($mpk', ek', \{x_i\}_{i \in [n]}$):</p> <p>Parse $ek' := \{(s_i, t_i)\}_{i \in [n]}$</p> <p>Parse $mpk' = (\mathbb{G}, g, h, c_d, \{h_i\}_{i \in [n]})$</p> <p>$r_e \leftarrow \mathbb{Z}_p$</p> <p>$ct = (g^r, h^r, \{g^{x_i} \cdot h_i^r\}_{i \in [n]})$</p>	<p>PAFE.Dec($mpk', \{w_i\}_{i \in [n]}, sk'_f, ct'$):</p> <p>Parse $mpk' = (\mathbb{G}, g, h, c_d, \{h_i\}_{i \in [n]})$</p> <p>Parse $sk'_f = (g^{\sum_{i=1}^n s_i \cdot w_i}, h^{\sum_{i=1}^n t_i \cdot w_i})$</p> <p>Parse $ct' = (g^r, h^r, \{g^{x_i} \cdot h_i^r\}_{i \in [n]})$</p> <p>$y = \frac{\prod_{i=1}^n g^{x_i \cdot h_i^r}}{g^{r \cdot (\sum_{i=1}^n s_i \cdot w_i)} \cdot h^{r \cdot (\sum_{i=1}^n t_i \cdot w_i)}}$</p> <hr/> <p>PAFE.ProveDec($mpk', \{w_i\}_{i \in [n]}, sk'_f, pk'_f, ct', y'$):</p> <p>Parse $mpk' = (\mathbb{G}, g, h, c_d, \{h_i\}_{i \in [n]})$</p> <p>Parse $sk'_f = (g^{\sum_{i=1}^n s_i \cdot w_i}, h^{\sum_{i=1}^n t_i \cdot w_i})$</p> <p>Parse $pk'_f = (g^{\sum_{i=1}^n s_i \cdot w_i}, h^{\sum_{i=1}^n t_i \cdot w_i})$</p> <p>Parse $ct' = (g^r, h^r, \{g^{x_i} \cdot h_i^r\}_{i \in [n]})$</p> <p>$\pi \leftarrow \text{NIZK.Prove}(mpk', \{w_i\}_{i \in [n]}, sk'_f, pk'_f, ct', y')$</p> <p>Return π</p> <hr/> <p>PAFE.VerifyDec($mpk', \{w_i\}_{i \in [n]}, pk'_f, ct', y', \pi'$):</p> <p>Parse $mpk' = (\mathbb{G}, g, h, c_d, \{h_i\}_{i \in [n]})$</p> <p>Parse $pk'_f = (g^{\sum_{i=1}^n s_i \cdot w_i}, h^{\sum_{i=1}^n t_i \cdot w_i})$</p> <p>Parse $ct' = (g^r, h^r, \{g^{x_i} \cdot h_i^r\}_{i \in [n]})$</p> <p>Return $\text{NIZK.Verify}(mpk', \{w_i\}_{i \in [n]}, pk'_f, ct', y', \pi')$</p>
---	--

Fig. 9. Publicly auditable MIFE for inner products with untrusted decryptor.

We now present our PA MIFE scheme that achieves public auditability in the presence of untrusted decryptors (PA-UD), using a non-interactive zero-knowledge argument (NIZK) [16] to produce a proof of correct decryption. Referring to the construction of Fig. 9, we observe that the decryptor needs only give the term $(g^r)^{sk_{f,g}} \cdot (h^r)^{sk_{f,h}}$ to potential auditors and convince them about the fact that sk_f has been used appropriately to generate this term. This can be done by proving discrete logarithm relations between what the verifier already knows and the information received from the decryptor, i.e., proving knowledge of a common discrete logarithm between multiple DDH tuples.

This can be done via a general-purpose NIZK or even a Σ -protocol for DDH tuples (e.g., see [32, Ch. 5.2]). In the second case, the resulting construction would be very efficient and would not require trusted CRS generation. To satisfy the security definition, the NIZK should be fully zero-knowledge; if instantiated with a Σ -protocol we assume the non-interactive version based on the Fiat-Shamir heuristic is used which also makes it zero-knowledge against arbitrary verifiers. Formally, the relation for the NIZK, in Camenisch-Stadler notation [18], is:

$$\mathbf{PK}\{(x, r) : (g, h, g^r, h^r, g^{\sum_{i=1}^n s_i \cdot w_i}, h^{\sum_{i=1}^n t_i \cdot w_i}, g^{r \cdot \sum_{i=1}^n s_i \cdot w_i}, h^{r \cdot \sum_{i=1}^n t_i \cdot w_i}, g^{u_d} \cdot h^{r_d})\}$$

We state the following theorem and delegate the proof for our publicly auditable MIFE scheme to the full version of our work [4].

Theorem 5. *The construction depicted in Fig. 9 is a PAFE scheme for inner-product functionalities. It is secure as per Definition 8 and is publicly auditable as per Definition 9, assuming that the MIFE scheme of [8] is IND-secure as per Definition 6 and the employed NIZK is computationally sound.*

8 Conclusion and Discussion

In this work we introduced public audibility in the context of functional encryption. We defined four flavors of PA, for each different corresponding corruption set among the participating parties, and presented corresponding constructions that achieve these definitions, as well as a novel indistinguishability security definition for PAFE. Our constructions rely on secure FE, commitments, and NIWI schemes to force the parties to prove their correct behavior. Finally, we proposed a multi-input PAFE scheme that supports linear combination functionalities expressed as vector inner-products. It builds upon previous MIFE schemes that produce “El Gamal”-style ciphertexts that are amenable to Σ -protocols, thus being potentially very efficient for use in practice.

Our work leaves many possible directions for future research in this topic. First, it would be of interest to design an adaptively secure PA-UEAD PAFE scheme, since our presented construction for this setting is only selectively secure. In this aspect, we believe it is possible to consider a different type of model relaxation of security, i.e., with “static” function key queries. This refers to an adversary who specifies mutually exclusive sets of functions for which it either requests functional secret keys or solely public ones. This setting is incomparable to selective security but it seems more applicable to several real-world applications like the ones we described in the introduction of the paper. Another direction would be to design and implement practical PAFE schemes in any of the auditability settings, at least for more expressive functionalities (e.g., quadratic functions from [12, 25]). Finally, it is worth exploring other use cases and applications for more efficient PAFE schemes, e.g., in the context of auditable cryptocurrencies [19, 27, 30] and auditable blockchain storage [33].

Acknowledgements. We would like to thank the anonymous reviewers for their constructive feedback. This work was partially supported by Hong Kong RGC under grant 16200721.

Appendix

A Proof of Theorem 1

We prove the PAFE security game indistinguishable regardless of the challenger bit. We define multiple Hybrids to go from the execution of the PAFE security

game with $b = 0$ to the execution with $b = 1$ and prove them subsequently indistinguishable. We state the advantage that the adversary has during each transformation and provide the total advantage at the end of our analysis.

Note that we exclude from our analysis adversarial strategies that trivially win the PAFE security game (by violating its winning conditions). This means that if the adversary issues a series of queries like (*) or (**) the advantage of the adversary is reduced to 0, from the PAFE security game (Definition 8).

$$\begin{array}{l|l} (*) : \text{QEnc}(x_0, x_1), \text{QSKeyGen}(f) & \text{such that} \\ (**) : \text{QEnc}(x_0, x_1) \rightarrow \text{ct}, \text{QPKeyGen}(f), \text{QDec}(\text{ct}, f) & f(x_0) \neq f(x_1) \end{array}$$

Now, observe that we can divide all remaining possible, non-trivially-winning, strategies into two cases, based on whether the adversary issues $\text{QDec}(\cdot, \cdot)$ queries (case (i)) or not (case (ii)).

Intuitively by making such a division first we “exploit” the fact that adversaries who do not issue $\text{QDec}(\cdot, \cdot)$ queries (case (ii)), essentially degenerate into FE-type adversaries. The only exception is that they can also have access to functional public keys (which are computationally hiding commitments). On the other hand, we know that the adversary in case (i) will issue at least one non-trivially-violating $\text{QDec}(\text{ct}, \cdot)$ query, for $\text{QEnc}(x_0, x_1) \rightarrow \text{ct}$. This allows us to define hybrids over the total number of QEnc queries that are subsequently different in just a single output of the $\text{QEnc}(x_0, x_1) \rightarrow \text{ct}^b$ query (based on the challenger bit) and prove them indistinguishable. In more detail, we present our analysis for the two cases below:

Proof (Security).

Case (i): We assume $\mathcal{A}_{\text{PAFE}}$ issues at least one $\text{QDec}(\cdot, \cdot)$ query. We prove indistinguishability of the game that $\mathcal{A}_{\text{PAFE}}$ plays when $b = 0$ and $b = 1$ through a series of hybrids. Below we define the hybrids and prove them consecutively indistinguishable. The challenger bit is represented in the game/hybrid exponents.

Hybrid \mathcal{G}_{UD}^0 : It is the security game when $b = 0$.

Hybrid $\mathcal{H}_{UD,1}^0$: It is exactly the same game as \mathcal{G}_{UD}^0 except for the computation of the c_d . In \mathcal{G}_{UD}^0 $c_d \leftarrow \text{Com.Commit}(\text{msk}, ; u_d)$, whereas in $\mathcal{H}_{UD,1}^0$ $c'_d \leftarrow \text{Com.Commit}(\top; u_d)$. From the hiding property of the employed commitment scheme no PPT adversary who sees a commitment can identify the committed value. Thus, $\mathcal{G}_{UD}^0 \approx \mathcal{H}_{UD,1}^0$ and more specifically, $\text{Adv}_{\mathcal{G}_{UD}^0 - \mathcal{H}_{UD,1}^0}^{\text{Distinguish}}(\mathcal{A}_{\text{PAFE}}) = \text{Adv}^{\text{Com-Hiding}}(\mathcal{A}_{\text{PAFE}})$.

Hybrid $\mathcal{H}_{UD,2}^0$: It is exactly the same game as $\mathcal{H}_{UD,1}^0$ except for the computation of π_d . In $\mathcal{H}_{UD,1}^0$ $\pi_d \leftarrow \text{NIWI}_d.\text{Prove}(\text{mpk}, \text{msk}, f, \text{sk}_f, r_f, \text{pk}_f, \text{ct}, y, c_d, u_d)$ using the first condition for relation $\text{R}_{UD,d}$, whereas in $\mathcal{H}_{UD,2}^0$, using the second condition of $\text{R}_{UD,d}$, $\pi'_d \leftarrow \text{NIWI}_d.\text{Prove}(\text{mpk}, \perp, f, \perp, \perp, \text{pk}_f, \text{ct}, y, c_d, u_d)$ respectively. From the witness indistinguishability property of NIWI_d no PPT adversary can distinguish

between which condition is satisfied for the generation of π_d . Thus, $\mathcal{H}_{UD,1}^0 \approx \mathcal{H}_{UD,2}^0$ and more specifically, $Adv_{\mathcal{H}_{UD,1}^0 - \mathcal{H}_{UD,2}^0}^{\text{Distinguish}}(\mathcal{A}_{\text{PAFE}}) = Adv_{\text{NIWI}}^{\text{WI}}(\mathcal{A}_{\text{PAFE}})$.

Hybrid $\mathcal{H}_{UD,3}^0$: It is exactly the same game as $\mathcal{H}_{UD,2}^0$ except for the computation of the y . In this case, we change y to be $y = f(x)$ instead of $y \leftarrow \text{FE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct})$. Remember that for $\mathcal{A}_{\text{PAFE}}$ to have non-negligible chance of winning in its game, it must be that for all functions f that $\mathcal{A}_{\text{PAFE}}$ issues a $\text{QSKeyGen}(f)$ query, for all $\text{ct} \leftarrow \text{QEnc}(x_0, x_1)$: $f(x) = f(x_0) = f(x_1)$. Additionally and similarly, for all functions f for which $\mathcal{A}_{\text{PAFE}}$ has issued $\text{QPKeyGen}(f)$ and $\text{QDec}(\text{ct}, f)$ queries, where $\text{ct} \leftarrow \text{QEnc}(x_0, x_1)$, it must be that $f(x) = f(x_0) = f(x_1)$. In any other case by the restrictions of the security game for PAFE $Adv^{\text{sec-PAFE}}(\mathcal{A}_{\text{PAFE}}(1^\lambda)) = 0$. Since $\mathcal{A}_{\text{PAFE}}$ cannot win in any of these two games with non-negligible advantage unless $f(x_0) = f(x_1)$, $\mathcal{H}_{UD,2}^0 \approx \mathcal{H}_{UD,3}^0$ and more specifically, $Adv_{\mathcal{H}_{UD,2}^0 - \mathcal{H}_{UD,3}^0}^{\text{Distinguish}}(\mathcal{A}_{\text{PAFE}}) = 0$.

Hybrid $\mathcal{H}_{UD,4}^0$: In this game we make the following change: the challenger samples $j' \leftarrow \{0, \dots, m+1\}$, initializes a counter $j = 0$, and when $\mathcal{A}_{\text{PAFE}}$ issues an encryption query, the challenger sets $j = j + 1$ and returns ct_j^b (we denote that query as $\text{QEnc}(x_{0,j}, x_{1,j})$, more concretely). Now, when $\mathcal{A}_{\text{PAFE}}$ issues a $\text{QPKeyGen}(f)$ query, \mathcal{C} checks whether $f(x_{0,j'}) \neq f(x_{1,j'})$. If so, it samples $z_f, r_f \leftarrow \mathbb{Z}_p$ and computes $\text{pk}_f \leftarrow \text{Com.Commit}(z_f, r_f)$. Remember that since $f(x_{0,j'}) \neq f(x_{1,j'})$ the adversary cannot issue a $\text{QSKeyGen}(f)$ or a $\text{QDec}(\text{ct}_{j'}^b, f)$ query — for that particular ciphertext. In such cases $\mathcal{A}_{\text{PAFE}}$ would trivially diminish its advantage to 0, contradicting our assumption that it has non-negligible advantage ϵ in winning the security game for PAFE. Therefore, from the hiding property of the underlying commitment scheme, similarly to $\mathcal{G}_{UD}^0 \approx \mathcal{H}_{UD,1}$, we get that $\mathcal{H}_{UD,3}^0 \approx \mathcal{H}_{UD,4}^0$ and more specifically, $Adv_{\mathcal{H}_{UD,3}^0 - \mathcal{H}_{UD,4}^0}^{\text{Distinguish}}(\mathcal{A}_{\text{PAFE}}) = Adv^{\text{Com-Hiding}}(\mathcal{A}_{\text{PAFE}})$.

Hybrid $\mathcal{H}_{UD,5,j}^b$: We now define a series of hybrids, indexed by j . In these hybrids we make the following change: the challenger samples $b \leftarrow \{0, 1\}$ and when $\mathcal{A}_{\text{PAFE}}$ issues a $\text{QEnc}(x_0, x_1)$ query \mathcal{C} returns $\text{ct}^0 \leftarrow \text{PAFE.Enc}(\text{mpk}, \text{ek}, x_0)$, if $j < j'$, $\text{ct}^1 \leftarrow \text{PAFE.Enc}(\text{mpk}, \text{ek}, x_1)$, if $j > j'$, and $\text{ct}^b \leftarrow \text{PAFE.Enc}(\text{mpk}, \text{ek}, x_b)$, if $j = j'$. Based on the choice of j we define $m+1$ sub-hybrids, which we denote by $\mathcal{H}_{UD,5,m+1}^b, \dots, \mathcal{H}_{UD,5,0}^b$. Clearly, $\mathcal{H}_{UD,4}^0 = \mathcal{H}_{UD,5,m+1}^b$, $\mathcal{H}_{UD,4}^1 = \mathcal{H}_{UD,5,0}^b$, and $\mathcal{H}_{UD,5,j}^1 = \mathcal{H}_{UD,5,j+1}^0$. Following we prove $\mathcal{H}_{UD,5,j}^0 \approx \mathcal{H}_{UD,5,j}^1$, which translates into $\mathcal{H}_{UD,5,j}^0 \approx \mathcal{H}_{UD,5,j+1}^0$, based on the above, and ultimately into $\mathcal{H}_{UD,4}^0 \approx \mathcal{H}_{UD,4}^1$.

Lemma 1. *Assuming the underlying FE scheme is secure as per Definition 4 $\mathcal{H}_{UD,5,j}^0 \approx \mathcal{H}_{UD,5,j}^1$.*

Proof. We prove this via contraposition. We construct an adversary \mathcal{A}_{FE} that utilizes $\mathcal{A}_{\text{PAFE}}$ to win in the security game of FE. Now, assuming $\mathcal{A}_{\text{PAFE}}$ issues at most m $\text{Qenc}(\cdot)$ queries, \mathcal{A}_{FE} functions as follows:

- Initialization: \mathcal{A}_{FE} receives mpk from \mathcal{C} , computes $\text{pp} \leftarrow \text{Com.Setup}(1^\lambda)$, samples $j^* \leftarrow \$_{[m]}$, initializes $\text{counter} = 0$, initializes a table \mathcal{T}_{enc} , samples $r_s \leftarrow \{0, 1\}^\lambda$, computes $c_d \leftarrow \text{Com.Commit}(\top; u_d)$, samples $b' \leftarrow \{0, 1\}$, and forwards the triple $(\text{pp}, \text{mpk}, c_d)$ to $\mathcal{A}_{\text{PAFE}}$.
- Encryption queries: When $\mathcal{A}_{\text{PAFE}}$ issues a $\text{QEnc}(x_0, x_1)$ query to \mathcal{A}_{FE} , the latter issues a $\text{QEnc}(x_j, x_j)$ query to \mathcal{C} and increments counter by 1. $x_j = x_0$ for $\text{counter} < j^*$, and $x_j = x_1$ for $\text{counter} > j^*$. For $\text{counter} = j^*$ \mathcal{A}_{FE} forwards the query to \mathcal{C} without any alteration. Regardless the case, \mathcal{C} returns a ciphertext ct , which \mathcal{A}_{FE} forwards to $\mathcal{A}_{\text{PAFE}}$.
- Functional secret key queries: When $\mathcal{A}_{\text{PAFE}}$ issues a QSKeyGen query to \mathcal{A}_{FE} , the latter forwards the query to \mathcal{C} , who responds with sk_f . \mathcal{A}_{FE} then checks if a QPKeyGen query has been issued for f . If not, it samples $r_f \leftarrow \$_{\{0, 1\}^\lambda}$ and computes $\text{pk}_f \leftarrow \text{Com.Commit}(\text{sk}_f; r_f)$, \mathcal{A}_{FE} forwards $(\text{sk}_f, \text{pk}_f)$ to $\mathcal{A}_{\text{PAFE}}$.
- Functional public key queries: When $\mathcal{A}_{\text{PAFE}}$ issues a $\text{QPKeyGen}(f)$ query to \mathcal{A}_{FE} , the latter checks whether $f(x_{0, j^*}) \neq f(x_{1, j^*})$. If so, \mathcal{A}_{FE} samples $r_f \leftarrow \$_{\{0, 1\}^\lambda}$, samples $z_f \leftarrow \$_{\{0, 1\}^\lambda}$, and computes $\text{pk}_f \leftarrow \text{Com.Commit}(z_f; r_f)$. Otherwise, \mathcal{A}_{FE} forwards a $\text{QSKeyGen}(f)$ query to \mathcal{C} , who responds with sk_f . \mathcal{A}_{FE} samples $r_f \leftarrow \$_{\{0, 1\}^\lambda}$, and computes $\text{pk}_f \leftarrow \text{Com.Commit}(\text{sk}_f; r_f)$. In any case \mathcal{A}_{FE} returns pk_f to $\mathcal{A}_{\text{PAFE}}$.
- Decryption queries: When $\mathcal{A}_{\text{PAFE}}$ issues a $\text{QDec}(\text{ct}, f)$ query to \mathcal{A}_{FE} , the latter assigns $y \leftarrow f(x_j)$ and $\pi_d \leftarrow \text{NIWI}_d.\text{Prove}(\text{mpk}, \top, f, \perp, \perp, \text{pk}_f, \text{ct}, y, c_d, u_d)$. \mathcal{A}_{FE} forwards (y, π_d) to $\mathcal{A}_{\text{PAFE}}$.
- Finalization: $\mathcal{A}_{\text{PAFE}}$ outputs a bit b' which \mathcal{A} forwards to \mathcal{C} .

The advantage \mathcal{A}_{FE} has in winning the FE IND-security game utilizing $\mathcal{A}_{\text{PAFE}}$ is $\frac{\epsilon}{m} > \text{negl}(\lambda)$. This derives from the fact that \mathcal{A}_{FE} needs to “guess” correctly the $\text{ct}_j^b \leftarrow \text{Qenc}(\cdot, \cdot)$ query for which $\mathcal{A}_{\text{PAFE}}$ will issue at least one “legitimate” $\text{QDec}(\cdot, \text{ct}_j^b)$ query; and does so by sampling j^* at random.

Thus, $\mathcal{H}_{UD,4}^0 = \mathcal{H}_{UD,5,m+1}^b \approx \mathcal{H}_{UD,5,0}^b = \mathcal{H}_{UD,4}^1$ and more specifically:
 $Adv_{\mathcal{H}_{UD,4}^0 - \mathcal{H}_{UD,4}^1}^{\text{Distinguish}}(\mathcal{A}_{\text{PAFE}}) = Adv^{\text{FE-IND security}}(\mathcal{A}_{\text{PAFE}})$.

Hybrid $\mathcal{H}_{UD,3}^1$: In this game we make the following change: When $\mathcal{A}_{\text{PAFE}}$ issues a $\text{QPKeyGen}(f)$ query, \mathcal{C} forwards $\text{pk}_f \leftarrow \text{PAFE.KeyGen}(\text{msk}, \text{mpk}, f)$ to $\mathcal{A}_{\text{PAFE}}$. From the hiding property of the underlying commitment scheme, similarly to $\mathcal{H}_{UD,3}^0 \approx \mathcal{H}_{UD,4}^0$, we get that $\mathcal{H}_{UD,4}^1 \approx \mathcal{H}_{UD,3}^1$ and more specifically,
 $Adv_{\mathcal{H}_{UD,4}^1 - \mathcal{H}_{UD,3}^1}^{\text{Distinguish}}(\mathcal{A}_{\text{PAFE}}) = Adv^{\text{Com-Hiding}}(\mathcal{A}_{\text{PAFE}})$.

Hybrid $\mathcal{H}_{UD,2}^1$: It is exactly the same game as $\mathcal{H}_{UD,3}^1$ except for the computation of the y . In this case, we change y to be $y \leftarrow \text{FE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct})$, instead of $y = f(x)$. Similarly to the case $\mathcal{H}_{UD,2}^0 \approx \mathcal{H}_{UD,3}^0$, we get that $\mathcal{H}_{UD,3}^1 \approx \mathcal{H}_{UD,2}^1$ and more specifically, $Adv_{\mathcal{H}_{UD,3}^1 - \mathcal{H}_{UD,2}^1}^{\text{Distinguish}}(\mathcal{A}_{\text{PAFE}}) = 0$.

Hybrid $\mathcal{H}_{UD,1}^1$: It is exactly the same game as $\mathcal{H}_{UD,2}^1$ except for the computation of π_d . In $\mathcal{H}_{UD,2}^0$ $\pi'_d \leftarrow \text{NIWI}_d.\text{Prove}(\text{mpk}, \perp, f, \perp, \perp, \text{pk}_f, \text{ct}, y, c_d, u_d)$ using

the second condition of $R_{UD,d}$, whereas in $\mathcal{H}_{UD,1}^0$, using the first condition for relation $R_{UD,d}$, $\pi_d \leftarrow \text{NIWI}_d.\text{Prove}(\text{mpk}, \text{msk}, f, \text{sk}_f, r_f, \text{pk}_f, \text{ct}, y, c_d, u_d)$. From the witness indistinguishability property of NIWI_d , similarly to $\mathcal{H}_{UD,1}^0 \approx \mathcal{H}_{UD,2}^0$ we get that $\mathcal{H}_{UD,2}^1 \approx \mathcal{H}_{UD,1}^1$ and more specifically, $\text{Adv}_{\mathcal{H}_{UD,2}^1 - \mathcal{H}_{UD,1}^1}^{\text{Distinguish}}(\mathcal{A}_{\text{PAFE}}) = \text{Adv}_{\text{NIWI}}^{\text{WI}}(\mathcal{A}_{\text{PAFE}})$.

Game \mathcal{G}_{UD}^1 : It is the security game when $b = 1$. It is exactly the same game as $\mathcal{H}_{UD,1}^1$ except for the computation of the c_d . In $\mathcal{H}_{UD,1}^1$ $c_d \leftarrow \text{Com.Commit}(\top; u_d)$, whereas in \mathcal{G}_{UD}^1 $c_d \leftarrow \text{Com.Commit}(\text{msk}; u_d)$. From the hiding property of the employed commitment scheme no PPT adversary who sees a commitment can identify the committed value. Thus, $\mathcal{H}_{UD,1}^1 \approx \mathcal{G}_{UD}^1$ and to be more specific, $\text{Adv}_{\mathcal{H}_{UD,1}^1 - \mathcal{G}_{UD}^1}^{\text{Distinguish}}(\mathcal{A}_{\text{PAFE}}) = \text{Adv}^{\text{Com-Hidding}}(\mathcal{A}_{\text{PAFE}})$.

Therefore, the overall advantage $\mathcal{A}_{\text{PAFE}}$ has in case (i): $\text{Adv}_{\mathcal{G}_{UD}^0 - \mathcal{G}_{UD}^1}^{\text{Distinguish}(i)}(\mathcal{A}_{\text{PAFE}}) \leq 4 \times \text{Adv}^{\text{Com-Hidding}}(\mathcal{A}_{\text{PAFE}}) + 2 \times \text{Adv}_{\text{NIWI}}^{\text{WI}}(\mathcal{A}_{\text{PAFE}}) + \text{Adv}^{\text{FE-IND security}}(\mathcal{A}_{\text{PAFE}})$.

Case (ii): We assume $\mathcal{A}_{\text{PAFE}}$ issues no $\text{QDec}(\cdot, \cdot)$ queries and has a non-negligible advantage ϵ in winning the PAFE security game. In this case we exploit the fact that $\mathcal{A}_{\text{PAFE}}$ will not issue a $\text{QSKeyGen}(f)$ query if there exists a pair of messages (x_0, x_1) in a $\text{QEnc}(x_0, x_1) \rightarrow \text{ct}$ query, such that $f(x_0) \neq f(x_1)$ and vice versa — since either way would trivially violate the winning conditions of the PAFE security game, rendering $\text{Adv}^{\text{sec-PAFE}}(\mathcal{A}_{\text{PAFE}}) = 0$ (see case (*)). We therefore can construct a “greedy” adversary \mathcal{A}'_{FE} who utilizes $\mathcal{A}_{\text{PAFE}}$ and wins the FE IND-security game with non-negligible advantage. \mathcal{A}'_{FE} forwards all queries made by $\mathcal{A}_{\text{PAFE}}$ to its challenger, except for $\text{QPKeyGen}(\cdot)$ ones. When $\mathcal{A}_{\text{PAFE}}$ issues a $\text{QPKeyGen}(f)$ query to \mathcal{A}'_{FE} , the latter checks whether $\exists \text{ct} \leftarrow \text{QEnc}(x_0, x_1)$ such that $f(x_0) \neq f(x_1)$. If so, \mathcal{A}'_{FE} samples $r_f \leftarrow \mathfrak{s}\{0, 1\}^\lambda$, samples $z_f \leftarrow \mathfrak{s}\{0, 1\}^\lambda$, and computes $\text{pk}_f \leftarrow \text{Com.Commit}(z_f; r_f)$. Otherwise, \mathcal{A}'_{FE} forwards a $\text{QSKeyGen}(f)$ query to \mathcal{C} , who responds with sk_f . \mathcal{A}'_{FE} samples $r_f \leftarrow \mathfrak{s}\{0, 1\}^\lambda$, and computes $\text{pk}_f \leftarrow \text{Com.Commit}(\text{sk}_f; r_f)$. In any case \mathcal{A}'_{FE} returns pk_f to $\mathcal{A}_{\text{PAFE}}$. Since the commitment scheme is computationally hiding \mathcal{A}'_{FE} has also $\epsilon > \text{negl}(\lambda)$ advantage in winning the FE IND-security game, violating our initial assumption.

(Auditability). We show that no PPT adversary $\mathcal{A}_{\text{PA-UD}}$ can violate the PA-UD property of PAFE, assuming a computationally sound NIWI for relation $R_{UD,d}$, NIWI_d and a perfectly binding commitment scheme Com . We examine two cases. First, there is the case where the adversary $\mathcal{A}_{\text{PA-UD}}$ may output a tuple T that satisfies $R_{UD,d}$. If so, it either satisfies the condition that ensures that PA-UD holds ($\text{pk}_f \leftarrow \text{Com}(\text{sk}_f; r_f) \wedge y \leftarrow \text{FE.Dec}(\text{mpk}, f, \text{sk}_f, \text{ct})$), or the “trapdoor” condition $c_d \leftarrow \text{Com}(\top; u_d)$. In the PA-UD setting c_d is generated by the authority (assumed to be honest in this setting), meaning that no malicious decryptor can generate a convincing proof using condition (2) of $R_{UD,d}$.

Otherwise, without loss of generality we distinguish between the following regarding the first condition: T either violates the commitment or the algorithmic condition. Since the commitment is perfectly binding, $\forall \text{pk}_f \nexists (\text{sk}_f^*, r_f^*) \neq (\text{sk}_f, r_f)$

such that $\text{pk}_f \leftarrow \text{Com}(\text{sk}_f^*; r_f^*) \wedge \text{pk}_f \leftarrow \text{Com}(\text{sk}_f; r_f)$. Additionally, since mpk and ct , are provided by trusted entities and the uniquely correct sk_f is used in the FE.Dec algorithm, y is also explicitly correct (due to the correctness of the underlying FE scheme). Due to the soundness property of NIWI_d any proof π^* that passes verification is generated for accepting PA-UD statements using valid witnesses. Therefore, no PPT $\mathcal{A}_{\text{PA-UD}}$ can break the PA-UD property with non-negligible advantage.

References

1. Delloite-US. <https://www2.deloitte.com/us/en.html>
2. Facebook Community Standards. <https://www.facebook.com/communitystandards>
3. KPMG-CN. <https://home.kpmg/cn/en/home.html>
4. Publicly auditable functional encryption. <https://cse.hkust.edu.hk/vkoutsos/pafe.pdf> (2023)
5. Abdalla, M., Benhamouda, F., Gay, R.: From single-input to multi-client inner-product functional encryption. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 552–582. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_19
6. Agrawal, S., Goyal, R., Tomida, J.: Multi-input quadratic functional encryption from pairings. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 208–238. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_8
7. Agrawal, S., Goyal, R., Tomida, J.: Multi-party functional encryption. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13043, pp. 224–255. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90453-1_8
8. Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 333–362. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_12
9. Ambrona, M., Fiore, D., Soriente, C.: Controlled functional encryption revisited: Multi-authority extensions and efficient schemes for quadratic functions. Proc. Priv. Enhancing Technol. **2021**(1), 21–42 (2021). <https://doi.org/10.2478/popets-2021-0003>
10. Badertscher, C., Kiayias, A., Kohlweiss, M., Waldner, H.: Consistency for functional encryption. In: 34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, 21–25 June 2021, pp. 1–16. IEEE (2021). <https://doi.org/10.1109/CSF51468.2021.00045>
11. Badrinarayanan, S., Goyal, V., Jain, A., Sahai, A.: Verifiable functional encryption. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 557–587. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_19
12. Baltico, C.E.Z., Catalano, D., Fiore, D., Gay, R.: Practical functional encryption for quadratic functions with applications to predicate encryption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 67–98. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_3
13. Barbosa, M., Farshim, P.: Delegatable homomorphic encryption with applications to secure outsourcing of computation. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 296–312. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27954-6_19

14. Bellare, M., Palacio, A.: GQ and Schnorr identification schemes: proofs of security against impersonation under active and concurrent attacks. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 162–177. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_11
15. Bitansky, N., Paneth, O.: ZAPs and Non-interactive witness indistinguishability from indistinguishability obfuscation. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 401–427. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_16
16. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 2–4 May 1988, Chicago, Illinois, USA, pp. 103–112. ACM (1988). <https://doi.org/10.1145/62212.62222>
17. Boneh, D., Sahai, A., Waters, B.: Functional encryption: definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_16
18. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0052252>
19. Chatzigiannis, P., Baldimtsi, F.: MINILEDGER: compact-sized anonymous and auditable distributed payments. In: Bertino, E., Shulman, H., Waidner, M. (eds.) ESORICS 2021. LNCS, vol. 12972, pp. 407–429. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88418-5_20
20. Chotard, J., Dufour-Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Dynamic decentralized functional encryption. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12170, pp. 747–775. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56784-2_25
21. Chotard, J., Dufour Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Decentralized multi-client functional encryption for inner product. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 703–732. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_24
22. Confessore, N.: Cambridge analytica and facebook: the scandal and the fallout so far. <https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html> (2018)
23. Gentry, C.: A fully homomorphic encryption scheme, Ph. D. thesis, Stanford University, USA (2009). <https://searchworks.stanford.edu/view/8493082>
24. Goldwasser, S., et al.: Multi-input functional encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 578–602. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_32
25. Gong, J., Qian, H.: Simple and efficient FE for quadratic functions. *Des. Codes Crypt.* **89**(8), 1757–1786 (2021). <https://doi.org/10.1007/s10623-021-00871-x>
26. Goyal, V., Jain, A., O’Neill, A.: Multi-input functional encryption with unbounded-message security. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 531–556. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_18
27. Kang, H., Dai, T., Jean-Louis, N., Tao, S., Gu, X.: FabZK: supporting privacy-preserving, auditable smart contracts in hyperledger fabric. In: DSN 2019, pp. 543–555. IEEE (2019). <https://doi.org/10.1109/DSN.2019.00061>
28. Koutsos, V., Papadopoulos, D., Chatzopoulos, D., Tarkoma, S., Hui, P.: Agora: a privacy-aware data marketplace. *IEEE Trans. Dependable Secur. Comput.* **19**(6), 3728–3740 (2022). <https://doi.org/10.1109/TDSC.2021.3105099>

29. Libert, B., T̃ițiu, R.: Multi-client functional encryption for linear functions in the standard model from LWE. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 520–551. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_18
30. Narula, N., Vasquez, W., Virza, M.: zkLedger: privacy-preserving auditing for distributed ledgers. In: Banerjee, S., Seshan, S. (eds.) NSDI 2018, pp. 65–80. USENIX Association (2018). <https://www.usenix.org/conference/nsdi18/presentation/narula>
31. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_27
32. Schoenmakers, B.: Cryptographic protocols. Lecture Notes, Department of Mathematics and Computer Science, Technical University of Eindhoven (2019)
33. Shafagh, H., Burkhalter, L., Hithnawi, A., Duquenois, S.: Towards blockchain-based auditable storage and sharing of IoT data. In: ACM CCSW@CCS 2017, pp. 45–50 (2017)
34. Soroush, N., Iovino, V., Rial, A., Roenne, P.B., Ryan, P.Y.A.: Verifiable inner product encryption scheme. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12110, pp. 65–94. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45374-9_3
35. Suzuki, T., Emura, K., Ohigashi, T., Omote, K.: Verifiable functional encryption using intel SGX. In: Huang, Q., Yu, Yu. (eds.) ProvSec 2021. LNCS, vol. 13059, pp. 215–240. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90402-9_12
36. Tomida, J.: Tightly secure inner product functional encryption: multi-input and function-hiding constructions. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 459–488. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_16

Advanced Primitives



Robustly Reusable Fuzzy Extractors in a Post-quantum World

Amit Deo^(✉) and Charlie Grover

Crypto Quantique, London, UK
{adeo,cgrover}@cryptoquantique.com

Abstract. We revisit the problem of robustly reusable fuzzy extractors (RRFEs) with post-quantum security. Our main focus is constructions secure in the quantum random oracle model (QROM) that can be built by modifying existing classical ROM constructions. To date, security in the QROM has not been considered in the context of RRFEs. More specifically, we achieve three core contributions. The first is to produce a simple QROM construction of a (non-reusable) robust fuzzy extractor with security bounds that do not depend explicitly on the number of correctable errors t . As Becker (ePrint/2017/493) showed, previous ROM proofs depend *heavily* on t , preventing their use in certain applications (e.g. to PUFs). Our second contribution is to produce the first RRFE with a security proof in the QROM. The security bounds here also do not depend explicitly on t . Importantly, the construction does not utilise random number generation which can be difficult to achieve on constrained devices in a PUF application. Finally, we suggest optimisations of the only existing post-quantum standard model RRFE capable of correcting a linear number of errors, showing that it is far less efficient than our QROM construction.

Keywords: post-quantum security · QROM · fuzzy extractors · robustness

1 Introduction

Fuzzy extractors [14] were introduced by Dodis et al. to enable secure derivation of cryptographic keys from entropic (but non-uniform) *noisy* data sources. Informally, a fuzzy extractor consists of two algorithms: a generation algorithm FE.Gen and a reproduce algorithm FE.Rep. The generation algorithm takes as input a piece of source data w from a noisy entropic source and outputs a “hint” associated to the data P along with a secret key k i.e. $(P, k) := \text{FE.Gen}(w)$. The reproduce algorithm takes as input a piece of data w' and hint P , and outputs a key $k' := \text{FE.Rep}(w', P)$. These algorithms must achieve:

- Correctness: If w and w' are close (e.g. at most Hamming distance t apart for binary data), it should always be the case that $k = k'$.

- Key uniformity: Given P , the key k is indistinguishable from a uniform key, assuming that w is drawn from a distribution with sufficient entropy.

The first property says that if w and w' are two noisy readings of the same source, the keys derived by the generation and reproduce algorithms are equal. The second property says that an attacker that knows the hint P cannot distinguish the key from uniform (assuming w comes from some high entropy distribution).

Applications of Fuzzy Extractors. The original application of fuzzy extractors envisaged by Dodis et al. was biometric authentication [14], where a secret key is produced by taking some entropic biometric reading e.g. iris scans. Reading biometric data is subject to inherent noise, so it is difficult to reproduce a single secret key. Fuzzy extractors solve this problem by storing a (public) hint associated to an initial biometric reading. Each follow-up read is combined with this hint to correctly rederive the original key. Additionally, fuzzy extractors transform non-uniform biometric data into *uniform* keys suitable for cryptographic use. A similar application is the derivation of secret keys from so called “Physical Unclonable Functions” (PUFs). A PUF is essentially a piece of light-weight hardware able to output a silicon fingerprint of itself that can be reproduced on-demand rather than stored in non-volatile memory. They are often used to as a “root-of-trust” in secure key generation on IoT devices [21] because they are easily implemented on integrated circuits. The silicon fingerprint is typically derived from small uncontrollable variations in the manufacturing process. For example, the well-known SRAM PUF [18] relies on manufacturing variations leading to fairly stable high-entropy 0/1 power-up patterns in SRAM cells. These random patterns can serve as silicon fingerprints. Unfortunately, the power-up pattern is not perfectly reproducible, and uncontrollable variations in temperature or supply voltage can cause bit flips when comparing repeated readings.

Robustness. Following the seminal paper of Dodis et al. many works have considered additional security properties of fuzzy extractors. For example, the idea of a *robust* fuzzy extractor was introduced by Boyen et al. [8]; robustness requires that if an honestly produced hint is overwritten with some potentially malicious value, the reproduce algorithm detects this and outputs a failure symbol \perp . This means that even if an adversary can overwrite the hint, it cannot force a device to use an incorrect key. The robust scheme of Boyen et al. is built from any secure sketch [14] which is a primitive consisting of a generation algorithm SS.Gen and a reconciliation mechanism SS.Rec . Informally, if $s = \text{SS.Gen}(w)$ and w' is a bit string differing from w in at most t positions, then $\text{SS.Rec}(w', s) = w$. For security it is required that the unpredictability of w (i.e. the min-entropy of w) given s remains high. From a *well-formed* secure sketch¹, the robust construction of Boyen et al. can be interpreted using a hash function H in the random oracle model (ROM) via the following description, where H is parsed as (H_1, H_2) :

¹ Well-formed roughly means that for any w and sketch value s , $\text{SS.Rec}(w, s)$ is within Hamming distance t of w , where t is the maximal number of correctable errors.

- FE.Gen(w) : Output hint $P = (s, h) := (\text{SS.Gen}(w), H_1(w, \text{SS.Gen}(w)))$ and key $k = H_2(w, s)$.
- FE.Rep($w', P = (s, h)$) : Compute $\tilde{w} := \text{SS.Rec}(w', s)$. If $H_1(\tilde{w}, s) \neq h$, output \perp . Else output $k' = H_2(\tilde{w}, s)$.

Intuitively, an adversary outputting a valid hint must know the value of (w, s) (or produce a hash pre-image) to obtain a correct hash value in FE.Rep. In the security argument, well-formedness of the secure sketch helps relate the unpredictability of \tilde{w} given s to the unpredictability of w' , and subsequently w . Unfortunately the method of proof leads to a security bound depending on $\text{Vol}_t = \|\{x : HW(x) \leq t\}\|$ where HW denotes Hamming weight. As analysed by Becker [5], this dependence on t restricts the utility of Boyen et al.’s scheme (and also that of [13]) in practical scenarios. Specifically, Becker shows that to achieve both correctness and security, the maximal error rate that can be corrected is 3% when BCH codes are used to implement SS. This error rate is significantly lower than a “typical” PUF rate of 15%. Therefore, a natural question is:

(Q1) *Can we obtain a security bound for the simple hash-based robust fuzzy extractor that is independent of Vol_t and valid in the (Q)ROM?*

Moving away from the ROM, there are many examples of robust fuzzy extractors whose security holds in the *standard model* such as those of [11, 13]. The first of these is particularly interesting in the context of robustness, as its performance nearly matches the that of the original non-robust construction of Dodis et al. Additionally, the security proof is information-theoretic. However, it is worth noting that the security proof is in a common reference string (CRS) model, and that the construction requires random number generation (which is not the case for the simple hash based constructions considered in this paper); reliance on a random number generator (RNG) can be problematic on constrained devices, such as the IoT devices relying on PUFs for key derivation.

Reusability. Another extension of the original security properties is the notion of *reusability* [7]. This notion considers the case where the generation algorithm is run multiple times to produce several hints/keys corresponding to noisy versions of the same source data. Reusability asks that distinct keys derived from a single noisy source remain uniform from the perspective of an adversary that has access to the corresponding hint values (and a selection of potentially compromised keys). This situation occurs when a single noisy source is used to derive multiple keys. Examples of reusable fuzzy extractors secure in the ROM can be found in [1, 7, 9] whereas schemes secure in the standard model can be found in [4, 9, 24, 25]. The notions of robustness and reusability were subsequently combined to define robustly reusable fuzzy extractors [26]. Examples of robustly reusable fuzzy extractors in the standard model can be found in [12, 26, 27]. Note that all robustly reusable fuzzy extractor constructions to date have been proven secure in the standard model using standard computational assumptions.

Post-quantum Constructions. It is well-known that many standard number theoretic assumptions used in cryptography do not hold against *quantum* capable adversaries. This led to the construction of schemes based on *post-quantum* assumptions that are believed to hold for *quantum* adversaries. Generally speaking, using the ROM can lead to relatively simple and very efficient constructions. However, the classical ROM is not appropriate for post-quantum security because quantum adversaries may have access to quantum implementations of the random oracle, and thus make superposition queries to it. This motivated the introduction of the quantum random oracle model [6] or QROM. To our knowledge, the problem of constructing fuzzy extractors in the QROM is yet to be studied. This leads to the important open question:

(Q2) *Are there practical, simple and efficient robustly reusable fuzzy extractors provably secure in the QROM?*

Looking now to the standard model, post-quantum fuzzy extractors have been considered extensively, often based on the learning with errors (LWE) assumption [4, 15, 19, 24]. Further, post-quantum robustly reusable fuzzy extractors are given in [12, 27]. Unfortunately, the first of these works cannot correct a linear number of errors (i.e. cases where the Hamming distance between w and w' is a constant fraction of their bit-lengths) rendering it unsuitable for applications such as PUFs where a linear error rate is expected. On the other hand, the work of [12] does not have this issue. A simple observation is that the scheme from [12] is based on plain LWE [22] (or more precisely on learning parity with noise (LPN) which may be conceptualised as LWE with modulus 2). Typically, converting schemes from LWE/LPN to ring-LWE/LPN [17, 20] saves a factor of $O(\lambda)$ (where λ is the security parameter) somewhere within the scheme. This saving arises because $O(\lambda) \times O(\lambda)$ matrices used in plain LWE/LPN are replaced by degree $O(\lambda)$ polynomials in ring-LWE/LPN. Therefore, a natural question is:

(Q3) *Can we improve on the efficiency of existing standard model robustly reusable fuzzy extractors by using the ring-LWE/LPN assumption?*

Contributions. Our contribution is to tackle the three open questions (Q1)-(Q3) outlined above. For (Q1), we prove that a lightly modified version of the hash-based scheme of Boyen et al. can be proved robust in the QROM with security bound independent of Vol_t according to the definition of [26]. In fact, this particular definition is slightly weaker than the strongest definition of Boyen et al. [8] as the source error between reads is assumed to arbitrarily depend on the hint seen by the adversary. In other words, the definition of [26] captures the case where natural source errors are independent of the true source value, as well as the case where the adversary is able to arbitrarily manipulate source errors. The stronger definition of Boyen et al. additionally considers arbitrary source error distributions that can even depend on the secret source value. Nonetheless, under the plausible assumption that source errors are independent of the true read value, our scheme and security proof are applicable. This assumption

is particularly suited to the PUF setting as elaborated next. For SRAM PUFs and more generally, the error profile between multiple reads is typically independent of the values of the PUF bits themselves. For instance, an SRAM PUF derives an output bit by considering the power-up behavior of an SRAM cell containing two cross-coupled and nominally identical inverters. Process variations cause mismatches between the inverters, so that the cell typically powers up in either the 0 or 1 state. However, the presence of electronic noise occasionally changes the powerup state, resulting in imperfect reliability. Since the cell is symmetric, the electronic noise, and thus error probability, will be independent of the direction of mismatch between the inverters. Briefly, there are two main modifications imposed on the previous hash-based construction: the first is that the requirement of well-formedness can be dropped, and the second is that the underlying secure sketch must satisfy a certain linearity property originally introduced in [11] in the context of standard model robust fuzzy extractors. To produce our QROM proof, we use a different strategy to the original proof of Boyen et al. allowing us to utilise known techniques from the QROM literature. In particular, we make use of a one-way to hiding (O2H) lemma [3, 23] and Zhandry’s quantum query recording techniques [28] to produce our QROM proof. Unlike in previous hash-based constructions, the security bound derived does not depend explicitly on t and therefore does not place strong restrictions on the number of correctable errors. Due to space requirements, the results with their concrete security bounds are stated as Theorem 1 and Theorem 2, but the proofs are deferred to Appendix E and the full version of this paper. The proof of the more general and main result of reusable robustness (Q2) is included in the main body instead.

Next we answer (Q2), showing that a minor modification of the robust construction is robustly reusable in the QROM (Theorem 3), giving concrete security bounds. The security game gives the adversary access to a reusability oracle that on input a perturbation Δ , with Hamming weight at most t , returns a hint and key computed at $w + \Delta$. Again, the definitions are from [26] and consider cases where source errors are adversarially chosen, so may depend arbitrarily on the hints seen but not on the source value as in [9]. We note that the only construction achieving the stronger reusability definition [9] is unable to handle adaptive reusability queries or a linear number of errors. Additionally, even optimisations of this construction [10] have hint sizes in the 100’s of MBs or more in practical applications. The only further modification required to the simple hash construction is that the underlying secure sketch must be linear, homomorphic *and* well-formed². The syndrome-based code offset secure sketch of Dodis et al. [14] possesses these three properties. The homomorphic property was previously considered in the context of standard model robustly reusable fuzzy extractors [26]. The security proofs here make crucial use of the same aforementioned results from the QROM literature, and the derived security bounds depend on the amount of entropy m' left in a source after leaking a secure sketch value, but

² The well-formed property can be dropped for the sake of weaker properties (see Remark 1), but we choose to use properties already defined in the literature.

not explicitly on t . However, the scheme’s efficiency still suffers somewhat when t is moderately large, as the number of required bits n grows relatively quickly with t in syndrome code-offset sketches; for details, see Sect. 4.3. Otherwise, the construction is very efficient, requiring just $n - \kappa + \lambda$ bits of hint storage (where n and κ are the length and dimension respectively of the code used in the syndrome code-offset sketch) to extract a large number of key bits. In short, κ may be as small as 4λ for robust reusability in the QROM and 2λ in the ROM. We note that there is an unavoidable implicit dependence on t , both due to the Singleton bound $2t \leq n - \kappa$ and the relation $n - \kappa = m - m'$ arising from the code-offset secure sketch, where m and m' are the min-entropies of the source distribution before and after learning a secure sketch value respectively. However, this weak dependence sharply contrasts the Vol_t dependence in previous works.

Finally, we answer (Q3) by translating the robustly reusable standard model construct of Cui et al. [12] from plain- to ring-LPN to improve efficiency. If the underlying ring is a field, the transformation is trivial. However, using non-field rings that split into many factors allows for fast multiplication algorithms (via e.g. fast Fourier transform-like methods [2]), so we describe a small modification of Cui et al. to permit the use of arbitrary rings. As we show in Sect. 5, the requirement on the error correcting code for the syndrome-based code offset secure sketch is *much* stricter than that of the QROM construction. This leads to a required number of PUF bits and hint storage that is orders of magnitude larger than the hash-based construction. The scheme also requires an RNG, unlike the QROM construction.

Roadmap. We begin with preliminaries in Sect. 2. We then present the simple hash-based construction in Sect. 3, followed by a list of *plain* robustness results proven in Appendix E and the full version of this paper. In Sect. 4, we prove reusability and *reusable* robustness in the QROM and include an account of parameter restrictions (Sect. 4.3). We modify the LPN-based robustly reusable fuzzy extractor of Cui et al. in Sect. 5 and show that the parameter restrictions are noticeably stricter than those arising in the QROM proofs. Finally, we conclude the paper, suggesting directions for future work in Sect. 6.

2 Preliminaries

We use standard mathematical set notation e.g. $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{C}$ and asymptotic notation $\omega(\cdot), O(\cdot), \text{poly}(\cdot)$ etc. We denote vectors using boldface font and consider vectors as strings by concatenating the individual entries of a vector. If \mathbf{v} is a vector over \mathbb{R} , then $\|\mathbf{v}\|$ denotes its Euclidean norm. Alternatively, if Δ is a *binary* vector/string, then $\|\Delta\|$ denotes the Hamming weight of Δ . A unitary matrix $U \in \mathbb{C}^{n \times n}$ satisfies $U^\dagger U = \mathbf{I}$ where U^\dagger denotes the Hermitian conjugate of the matrix U and \mathbf{I} is the $n \times n$ identity matrix. For any $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, 2, \dots, n\}$. We will denote the security parameter as λ . An algorithm is said to be efficient or polynomial time if it runs in time $\text{poly}(\lambda)$. Take any random variable X and denote its distribution as \mathcal{D}_X . We write $x \leftarrow \mathcal{D}_X$

to denote that x is sampled from the distribution \mathcal{D}_X . We also write $x \leftarrow X$ to mean the same thing as $x \leftarrow \mathcal{D}_X$ by abuse of notation. Also, if X is a set, $x \leftarrow X$ means that x is sampled uniformly from the set X . If X is a random variable, then for any algorithm \mathcal{A} , the output of the algorithm on input X , i.e. $\mathcal{A}(X)$ is also a random variable. We use the notation $\Pr_{x \leftarrow X}[y \leftarrow \mathcal{A}(x)]$ to denote the probability that $\mathcal{A}(X)$ takes the value y . When an algorithm has a superscript, it means that the algorithm has “black-box” oracle access to the function present in the superscript. For example, \mathcal{A}^H indicates that the algorithm has oracle access to some function H . The min-entropy of a random variable X is defined as $\tilde{H}_\infty(X) := -\log(\max_x \Pr[X = x])$. Take a secondary random variable Y with distribution D_Y . Note that the event $Y = y$ induces a marginal distribution on X . The conditional min-entropy of X given Y is then defined as $\tilde{H}_\infty(X | Y) := -\log(\mathbb{E}_{y \leftarrow Y}[\max_x \Pr[X = x | Y = y]])$.

2.1 Quantum Random Oracle Model (QROM)

For the basics of quantum algorithms/computation, see Appendix A. We now discuss quantum algorithms with quantum access to a random oracle H . Since quantum computation is represented by unitary operators/matrices, we need that queries to H change the state of an adversary in a unitary way. H is usually implemented as an oracle that performs the following transformation $H : |x, y, z\rangle \mapsto |x, y \oplus H(x), z\rangle$ where z is a place-holder for all registers not affected by random oracle queries. Slightly abusing notation so that H denotes the unitary and the function itself, the quantum computation of an algorithm takes the form $U_q H U_{q-1} H \dots U_2 H U_1 |\varphi_0\rangle$ where $|\varphi_0\rangle$ is an initial state and the U_i 's are unitaries. In order to obtain the output of a quantum algorithm, we then take a measurement of the output registers.

Below is the simplest version of the one-way to hiding (O2H) lemma [23]. Note that the lemma allows algorithm \mathcal{A}^H to submit batches of queries in parallel to H and the result depends on the *depth* of \mathcal{A}^H 's queries, i.e. the number of batches that \mathcal{A}^H sends to H . The formulation of the version of the O2H lemma below is from [3]. Using the more complex semi-classical oracle techniques of [3] does not appear to improve our final security bounds in any meaningful way, so we stick to the formulation below.

Lemma 1 (O2H Lemma). *Let X and Y denote input and output spaces respectively. Take (G, H, S, z) to follow some arbitrary distribution where $S \subseteq X$, $G : X \rightarrow Y$, $H : X \rightarrow Y$ such that $\forall x \in S, G(x) = H(x)$ and z belongs to the input space of a query-depth d quantum oracle algorithm \mathcal{A}^H . Define the algorithm $\mathcal{B}^H(z)$ as follows:*

- Sample $i \leftarrow \{1, \dots, d\}$
- Run $\mathcal{A}^H(z)$ until just before its i -th query to H
- Measure the oracle input registers and output the result of the measurement denoted T

Define $\text{Adv}(\mathcal{B}) := \Pr[S \cap T \neq \emptyset : T \leftarrow \mathcal{B}^G(z)]$. Then for any classical event E

1. $|\Pr[E : \mathcal{A}^H(z)] - \Pr[E : \mathcal{A}^G(z)]| \leq 2d \cdot \sqrt{\text{Adv}(\mathcal{B})}$.
2. $|\sqrt{\Pr[E : \mathcal{A}^H(z)]} - \sqrt{\Pr[E : \mathcal{A}^G(z)]}| \leq 2d \cdot \sqrt{\text{Adv}(\mathcal{B})}$.

We will also use a quantum query recording lemma from [28]. In this work, Zhandry introduced the “compressed standard oracle” denoted CStO that essentially adds the random oracle points queried by an adversary to some database registers D (that is itself a superposition). If a pair $(x, y) \in D$, then we write $D(x) = y$. The useful thing about the CStO oracle is that it almost preserves the output distribution of any algorithm when it is used in place of a random oracle. The particular quantum query recording lemma from [28] we use follows.

Lemma 2. *Suppose there is an algorithm \mathcal{A} that on input z and oracle access to random oracle $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ outputs tuples $(x_1, \dots, x_k, y_1, \dots, y_k, \zeta)$. Let R be some collection of such tuples. Suppose that on input z , \mathcal{A} outputs a tuple that belongs to R such that for $i = 1, \dots, k, H(x_i) = y_i$ with probability p . Now consider running \mathcal{A} on input z with respect to CStO and measuring the database D after \mathcal{A} produces its output. Let p' be the probability that on input z , \mathcal{A} 's output tuple is in R and $D(x_i) = y_i$ where $y_i \neq \perp$ for $i = 1, \dots, k$. Then $\sqrt{p} \leq \sqrt{p'} + \sqrt{k/2^n}$.*

2.2 Fuzzy Extractor Definitions

Secure Sketches. Secure sketches are a fundamental building block used to construct fuzzy extractors [14]. Intuitively, a secure sketch allows one to recover a secret value w from a nearby secret value w' while controlling the amount of leakage on these secret values.

Definition 1. *A (m, m', t) -secure sketch over a metric space \mathcal{M} with metric d is a pair of potentially randomised procedures $\text{SS.Gen} : \mathcal{M} \rightarrow \{0, 1\}^*$ and SS.Rec that satisfy the following properties:*

1. (Correctness) $\forall w, w' \in \mathcal{M}$ s.t. $d(w, w') \leq t, \text{SS.Rec}(w', \text{SS.Gen}(w)) = w$.
2. (Secrecy) \forall distributions W s.t. $\tilde{H}_\infty(W) \geq m, \tilde{H}_\infty(W | \text{SS.Gen}(W)) \geq m'$.

Intuitively, a (m, m', t) -secure sketch allows us to correct up to t errors for a min-entropy m input whilst retaining at least m' bits of min-entropy on leaking the secure sketch value. We also recall definitions of further properties next:

Definition 2 (Linear secure sketches [11]). *Suppose \mathcal{M} is also a group with an addition operation. A secure sketch SS is linear if \exists an efficiently computable function f s.t. for arbitrary (w, s^*, Δ) with $d(\Delta, 0) \leq t$, we have $\text{SS.Rec}(w + \Delta, s^*) = w + f(\Delta, s^*, s)$ where $s := \text{SS.Gen}(w)$.*

Definition 3 (Homomorphic secure sketches [26]). *A secure sketch SS is homomorphic if $\forall w_1, w_2 \in \mathcal{M}, \text{SS.Gen}(w_1 + w_2) = \text{SS.Gen}(w_1) \oplus \text{SS.Gen}(w_2)$.*

Definition 4 (Well-formed secure sketches [8]). *A (m, m', t) secure sketch SS is well-formed if $\forall w' \in \mathcal{M}$ and $s, d(\text{SS.Rec}(w', s), w') \leq t$.*

Note that one could also consider well-formed sketches where SS.Rec can output a failure symbol \perp . However, this would not interact well with the definition of linear secure sketches, so we do not consider such well-formedness definitions.

Example 1 (Syndrome Code-Offset [14]). For a linear $[n, k, d = 2t + 1]$ code $C \in \mathbb{Z}_2^{k \times n}$ with parity check matrix $H \in \mathbb{Z}_2^{n \times (n-k)}$, the (tweaked) syndrome code-offset secure sketch denoted SynSS is comprised of the following algorithms:

- $\text{SynSS.Gen}(w)$: For $w \in \{0, 1\}^n$, output $s = w \cdot H$.
- $\text{SynSS.Rec}(w', s)$: For any $s \in \{0, 1\}^{n-k}$ and $w' \in \{0, 1\}^n$, compute $\tilde{e} = s \oplus (w' \cdot H)$ and syndrome decode to compute e such that $e \cdot H = \tilde{e}$ and $\|e\| \leq t$. If such an e is not found, return w' . Otherwise, output $w' \oplus e$.

SynSS is a (m, m', t) secure sketch for $m' = m - (n - k)$. It clearly possesses the homomorphic and well-formed properties. For the linearity property, note that $\text{SynSS.Rec}(w \oplus \Delta, s^*)$ is $w \oplus (\Delta \oplus e)$ where e is a function of Δ, s^* and $w \cdot H =: s$.

Fuzzy Extractors. We now write the definition of a fuzzy extractor [14], altered to allow for security with respect to different classes of adversary. The original definition only considered statistical security meaning that the indistinguishability was written in terms of statistical distance. However, we generalise this to consider e.g. ROM/QROM adversaries.

Definition 5. For any metric space \mathcal{M} with metric d , positive reals m, ℓ, ϵ and positive integer t , a $(\mathcal{M}, m, \ell, t, \epsilon)$ fuzzy extractor with respect to adversary class \mathcal{A} is a collection of algorithms $(\text{FE.Gen}, \text{FE.Rep})$ such that

1. For any $w \in \mathcal{M}$, $\text{FE.Gen}(w)$ outputs a hint or “helper data string” P and a key string $R \in \{0, 1\}^\ell$.
2. FE.Rep takes any $w' \in \mathcal{M}$ and hint P as inputs and outputs a key-string or failure symbol \perp .
3. (Correctness) For any w, w' where $d(w, w') \leq t$, if $(P, R) \leftarrow \text{FE.Gen}(w)$, then $\text{FE.Rep}(w', P) = R$.
4. (Security/Uniformity) Consider any adversary \mathcal{A} in class \mathcal{A} and distribution \mathcal{W} with $\tilde{H}_\infty(\mathcal{W}) \geq m$. Then the advantage that \mathcal{A} has when distinguishing (P, U) from (P, R) where $(P, R) \leftarrow \text{FE.Gen}(\mathcal{W})$ and $U \leftarrow \{0, 1\}^\ell$ is at most ϵ . In other words, $\forall \mathcal{A} \in \mathcal{A}, \frac{1}{2} |\Pr[1 \leftarrow \mathcal{A}(P, R)] - \Pr[1 \leftarrow \mathcal{A}(P, U)]| \leq \epsilon$.

We can modify the above to define a fuzzy extractor in the common reference string (CRS) model. This is done by adding a procedure FE.Init that samples a public CRS that is used as additional input to the FE.Gen and FE.Rep algorithms. This CRS may not be tampered with in any of the fuzzy extractor definitions considered and the distribution of \mathcal{W} is assumed to be independent of the CRS.

Plain Robustness. We now introduce the plain robustness definition. Very informally, robustness asks that whenever a adversarially chosen hint P^* is used, the FE.Rep algorithm detects this and outputs the failure symbol \perp . The security definition allows an adversary to choose the shift Δ applied to the original source

value w that will be used to check the validity of the hint P^* . If the adversary class is computationally unbounded, then the distribution of Δ could depend arbitrarily on any of the information that the adversary knows e.g. the genuine hint P that is derived from w , but not directly on w itself (as in the stronger definition of [8]). Nonetheless, the definition below follows the more recent one used in [12,26,27] and is practically meaningful.

Definition 6. *Take any metric space \mathcal{M} with metric d , positive reals $m, \ell, \epsilon, \delta$ and positive integer t . A $(\mathcal{M}, m, \ell, t, \epsilon)$ fuzzy extractor has robustness δ with respect to adversary class \mathbf{A} if $\forall \mathcal{A} \in \mathbf{A}$ and distributions \mathcal{W} such that $\tilde{H}_\infty(\mathcal{W}) \geq m$, the following experiment outputs success with probability at most δ :*

- \mathcal{A} is given P computed as $(P, R) = \text{FE.Gen}(w)$ for some secret $w \leftarrow \mathcal{W}$.
- \mathcal{A} outputs (Δ, P^*) . The experiment outputs success if and only if $\|\Delta\| \leq t$, $P^* \neq P$ and $\text{FE.Rep}(w + \Delta, P^*) \neq \perp$.

The reader may notice that the challenger does not send R to the adversary in the above definition. Such scenarios are known as *pre-application* robustness. The analogous *post-application* robustness definition gives R to the adversary and is stronger. Note that the reusable robustness definition below implies the *post-application* version of robustness above.

Reusable Robustness. We now state the definition of a robustly reusable fuzzy extractor [26]. Intuitively, the robustly reusable property of a fuzzy extractor guarantees that both robustness and uniformity hold, even when an adversary can obtain *multiple* hints and some keys derived from noisy versions of a fixed source value. The definition will make use of two separate security experiments – one for reusability/uniformity and the other for reusable robustness (which is slightly different to the experiment in Definition 6). Each experiment gives the adversary access to a single oracle. For the reusability experiment the adversary is given *classical* access to the oracle $\mathcal{O}_b(\cdot)$. On adversarial input Δ , this oracle returns \perp if $\|\Delta\| > t$. Otherwise, it samples $(P, R) \leftarrow \text{Gen}(w + \Delta)$ and uniform U , returning (P, R) if $b = 0$ and (P, U) if $b = 1$. Note that choices of U are recorded and used again in the event of a repeated query. We may now describe the reusability experiment with bit b as $\text{Exp}_{\mathcal{A}}^{\text{re},b}(\lambda)$:

- Challenger \mathcal{C} samples $w \leftarrow \mathcal{W}$.
- Given oracle access to $\mathcal{O}_b(\cdot)$, \mathcal{A} outputs a bit i.e. $b' \leftarrow \mathcal{A}^{\mathcal{O}_b}()$.
- The output of the experiment is the bit b' .

We next describe the (reusable) robustness experiment. This involves a query list \mathcal{Q} and an oracle $\mathcal{O}_{\text{Gen}}(\cdot)$. On *classical* adversarial input Δ , this oracle returns \perp if $\|\Delta\| > t$ and returns $(P, R) \leftarrow \text{Gen}(w + \Delta)$ otherwise (adding P to the list \mathcal{Q}). The reusable robustness experiment $\text{Exp}_{\mathcal{A}}^{\text{ro}}(\lambda)$ can now be described as:

- Challenger \mathcal{C} samples $w \leftarrow \mathcal{W}$ and initialises $\mathcal{Q} = \emptyset$.
- $(P^*, \Delta^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{Gen}}(\cdot)}()$.
- **Output:**

- If $\|\Delta^*\| > t$, the experiment outputs 0.
- If $P^* \in \mathcal{Q}$ the experiment outputs 0.
- If $\text{Rep}(w + \Delta^*, P^*) = \perp$, the experiment outputs 0.
- Otherwise the experiment outputs 1.

Definition 7. A $(\mathcal{M}, m, \ell, t, \epsilon)$ fuzzy extractor has reusability ϵ_1 and reusable robustness ϵ_2 , and is therefore a $(\mathcal{M}, m, \ell, t, \epsilon, \epsilon_1, \epsilon_2)$ robustly reusable fuzzy extractor with respect to adversary class \mathbf{A} , if it satisfies the following two requirements:

1. (Reusability) For any adversary $\mathcal{A} \in \mathbf{A}$,

$$\frac{1}{2} \cdot \left| \Pr[\text{Exp}_{\mathcal{A}}^{\text{re},0}(\lambda) = 1] - \Pr[\text{Exp}_{\mathcal{A}}^{\text{re},1}(\lambda) = 1] \right| \leq \epsilon_1.$$

2. (Reusable Robustness) For any adversary $\mathcal{A} \in \mathbf{A}$,

$$\Pr[\text{Exp}_{\mathcal{A}}^{\text{ro}}(\lambda) = 1] \leq \epsilon_2.$$

3 Simple Hash Construction and Results

We will show that the following construction, built from a secure sketch SS over a space \mathcal{M} of size 2^η , will be a robust and also a robustly reusable fuzzy extractor according to the definitions from Sect. 2. We also assume that secure sketch values lie in a set \mathcal{S} whose size is 2^ν . The proofs will assume that $\mathcal{M} = \{0, 1\}^\eta$ and $\mathcal{S} = \{0, 1\}^\nu$ where $\eta, \nu \in \mathbb{Z}$ for simplicity, but may be easily generalised. The simple hash-based construction (denoted HFE) that we consider is essentially that of Boyen et al. [8] and uses a single random oracle $H : \{0, 1\}^\eta \times \{0, 1\}^\nu \rightarrow \{0, 1\}^{k+\ell}$ whose output is parsed into a k bit string and an ℓ bit string. The construction HFE is as follows:

- HFE.Gen(w) : Compute $s = \text{SS.Gen}(w)$, $(h_1, h_2) = H(w, s)$ where $h_1 \in \{0, 1\}^k$ and $h_2 \in \{0, 1\}^\ell$. Output randomness $R = h_2$ and hint $P = (s, h_1)$.
- HFE.Rep($\tilde{w}, (s, h_1)$) : Compute $w' = \text{SS.Rec}(\tilde{w}, s)$ and $(h'_1, h'_2) = H(w', s)$ where $h'_1 \in \{0, 1\}^k$ and $h'_2 \in \{0, 1\}^\ell$. If $h'_1 \neq h_1$ then output \perp . Otherwise, output h'_2 .

Note that we could divide H into two separate random oracles (with output lengths k and ℓ) but choose not to in order to ease notation. Another important point is that the extracted key-length ℓ may take any desired value as long as the bounds in the security theorem offer λ -bit security. This is a by-product of relying on random oracle models. We now state two new plain robustness results, the first of which is proven in Appendix E. The proof of the second can be found in the full version of this paper. In the next section, we state and prove the more involved theorem (Theorem 3) on reusable robustness in the QROM. In fact, Theorem 3 implies Theorems 1 and 2.

Theorem 1. *Let (SS, Rec) be a linear (m, m', t) secure sketch. Then the hash construction HFE is a robust $(\mathcal{M}, m, \ell, t, \epsilon = 2q \cdot 2^{-m'})$ fuzzy extractor with robustness $\delta \leq 2^{-k} + 2q \cdot 2^{-m'}$ in the classical ROM against unbounded adversaries making at most q queries to the random oracle.*

Theorem 2. *Let (SS, Rec) be a linear (m, m', t) secure sketch. Then the hash construction HFE is a robust $(\mathcal{M}, m, \ell, t, \epsilon = 2q \cdot 2^{-m'/2})$ fuzzy extractor with robustness*

$$\delta \leq \min \left\{ \begin{array}{l} q \cdot 2^{-m'} + 2(q+1) \cdot 2^{-m'/2} + 2\sqrt{q} \cdot 2^{-(k+m')/2} + 2^{-k}, \\ \left(\sqrt{q \cdot 2^{-m'}} + \sqrt{1/2^k} + 2\sqrt{q(q+1) \cdot 2^{-m'}} \right)^2 \end{array} \right\}$$

in the QROM against unbounded quantum adversaries making at most q queries to the quantum random oracle.

4 Simple Robustly Reusable Fuzzy Extractors

4.1 Reusability in the QROM

We now look to prove the reusability property of HFE. An interesting observation is that considering the proof of the following lemma, it turns out that the reusability condition would hold (with the same security bounds) even if the adversary is given *quantum* access to the reusability oracle.

Lemma 3. *Let SS be a homomorphic (m, m', t) secure sketch. Then HFE is a $(\mathcal{M}, m, \ell, t, \epsilon = 2q \cdot 2^{-m'/2})$ reusable fuzzy extractor with reusability*

$$\epsilon_1 \leq 2q \cdot 2^{-m'/2}$$

in the QROM with respect to unbounded quantum adversaries making at most q quantum random oracle queries.

Proof. Consider any distribution \mathcal{W} with $\tilde{H}_\infty(\mathcal{W}) \geq m$ and $w \leftarrow \mathcal{W}$. By the homomorphic property of the secure sketch, if we write $s = \text{SS.Gen}(w)$, then $s_\Delta := \text{SS.Gen}(w + \Delta) = s + \text{SS.Gen}(\Delta)$. The adversary \mathcal{A} (which we assume to be deterministic and unbounded) in the reusability experiment has classical access to \mathcal{O}_b and quantum access to the random oracle H used in the construction (which we parse as H_1 and H_2 by splitting the output H into the first k bits and the remaining ℓ bits). We also introduce a further independent random oracle H_3 with the same input/output spaces as H_2 to use in the proof. We can then say that on input Δ , \mathcal{O}_b returns

- $(s_\Delta, H_1(w + \Delta, s_\Delta), H_2(w + \Delta, s_\Delta))$ if $b = 0$
- $(s_\Delta, H_1(w + \Delta, s_\Delta), H_3(w + \Delta, s_\Delta))$ if $b = 1$

Now considering that \mathcal{A} is unbounded, we can simplify things by giving \mathcal{A} responses to all possible \mathcal{O}_b queries as part of its input. In fact, we do not even have to specify the s_Δ 's as they are computable from s and the \mathcal{O}_b oracle input Δ . Suppose, we include the values $z_{b=0} := \{s, (H(w + \Delta, s_\Delta))_\Delta\}$ where Δ ranges over all error vectors such that $\|\Delta\| \leq t$, as part of its input. Then for any Δ' , \mathcal{A} can compute $s_{\Delta'}$ (using the homomorphic property) and retrieve $H(w + \Delta', s_{\Delta'})$ itself from $z_{b=0}$ i.e. it can perfectly compute the responses of a classical oracle $\mathcal{O}_{b=0}$. Alternatively, if we include $z_{b=1} := \{s, (H_1(w + \Delta, s_\Delta), H_3(w + \Delta, s_\Delta))_\Delta\}$ as input, it would perfectly compute the responses to the $\mathcal{O}_{b=1}$ oracle. Therefore, we replace classical access to an \mathcal{O}_b oracle by including the appropriate extra information z_b in \mathcal{A} 's input.

We will be using the O2H lemma. In particular, let G be equal to H on all points apart from on the set $S := \{(w + \Delta, s_\Delta) : \|\Delta\| \leq t\}$. On all remaining points, G is uniformly chosen from $\{0, 1\}^{k+\ell}$. We consider first the case where the reusability experiment uses the bit $b = 0$. Then for $w \leftarrow \mathcal{W}$ and (G, H) sampled as above, the input of \mathcal{A} will be given as $z_{b=0} = (s, (H(w + \Delta, s_\Delta))_\Delta)$. Then Lemma 1 tells us that

$$|\Pr[\mathcal{A}^H(z_{b=0}) = 1] - \Pr[\mathcal{A}^G(z_{b=0}) = 1]| \leq 2q\sqrt{\Pr[\mathcal{B}^G(z_{b=0}) \in S]}$$

where \mathcal{B}^G is described in Lemma 1. Note that if \mathcal{B}^G 's output (w^*, s^*) is in S , then $w^* = w + \Delta$ and $s^* = s + \text{SS.Gen}(\Delta)$ for some Δ such that $\|\Delta\| \leq t$, which implies that $w^* - \text{Rec}(0, s^* - s) = w + \Delta - \Delta = w$. In other words, there is an algorithm $\bar{\mathcal{B}}^G$ that finds w on input $z_{b=0}$ with probability $\Pr[\mathcal{B}^G(z_{b=0}) \in S]$. Note that the input $z_{b=0}$ which is just s along with a set of uniform values that are entirely independent of the random oracle G . Therefore, the only w -dependent information that $\bar{\mathcal{B}}^G(z_{b=0})$ has access to is s implying that $\Pr[\mathcal{B}^G(z_{b=0}) \in S] = \Pr[\bar{\mathcal{B}}^G(z_{b=0}) = w] \leq 2^{-m'}$ due to the fact that SS is a (m, m', t) secure sketch. Therefore, $|\Pr[\mathcal{A}^H(z_{b=0}) = 1] - \Pr[\mathcal{A}^G(z_{b=0}) = 1]| \leq 2q \cdot 2^{-m'/2}$.

We can reuse the analysis above using the input $z_{b=1}$ to conclude that

$$|\Pr[\mathcal{A}^H(z_{b=1}) = 1] - \Pr[\mathcal{A}^G(z_{b=1}) = 1]| \leq 2q \cdot 2^{-m'/2}$$

using the same G and H . Also, the joint distributions of $(G, z_{b=0})$ and $(G, z_{b=1})$ are identical, so $\Pr[\mathcal{A}^G(z_{b=1}) = 1] = \Pr[\mathcal{A}^G(z_{b=0}) = 1]$. Putting everything together and using the triangle inequality, we have

$$\frac{1}{2} \cdot |\Pr[\mathcal{A}^H(z_{b=0}) = 1] - \Pr[\mathcal{A}^H(z_{b=1}) = 1]| \leq 2q \cdot 2^{-m'/2}.$$

□

4.2 Reusable Robustness in the QROM

We now consider proving the reusable robustness of the simple construction HFE in the QROM (Theorem 3 below). For more details on efficiency and parameter settings, see Sect. 4.3. However, we note here that similarly to the QROM

robustness result of Theorem 2, taking $k = \lambda$, $q = 2^\lambda$ enforces that we use a (m, m', t) -secure sketch with $m' \approx 4\lambda$ if we want λ -bit security in the QROM. In slightly more detail, the reusable robustness bound alone (i.e. ϵ_2) would allow the use of $m' \approx 3\lambda$, but the indistinguishability from uniform bounds (i.e. parameter ϵ, ϵ_2) are stricter, enforcing the choice $m' \approx 4\lambda$.

Recall that the reusable robustness experiment gives the adversary *classical* access to the \mathcal{O}_{Gen} oracle and that the adversary wins only if the Δ^* it returns was not queried to \mathcal{O}_{Gen} . This can be conceptualised by considering a quantum adversary that takes a measurement of a register, records the result and then submits the classical value to the \mathcal{O}_{Gen} oracle whenever it wants to make an oracle query. Note that this complication does not appear in the reusability proof of Lemma 3 as queries need not be recorded. To make our adversary more compatible with the O2H lemmas and standard quantum computation (i.e. to defer all measurements to the end of the computation), we may w.l.o.g. consider an adversary with *quantum access* to \mathcal{O}_{Gen} subject to the following:

- Every time the adversary queries \mathcal{O}_{Gen} , it first copies the contents of the \mathcal{O}_{Gen} query register into a database register D_{Gen} . Note that D_{Gen} will have multiple slots, and on the i -th query, the i -th slot will be filled in with the contents of the query register just before the i -th query to \mathcal{O}_{Gen} . The D_{Gen} query registers will therefore be in superposition.
- Just before performing the measurement to obtain the final output, measure the database register D_{Gen} and interpret the result as the list of classical queries that the adversary made to \mathcal{O}_{Gen} .

The joint distribution of the database measurement D_{Gen} and adversary state is identical to the distribution of the intermediate measurements and adversary state. For a detailed proof of this, see Appendix D. In this way, we can include the adversary's \mathcal{O}_{Gen} queries in D_{Gen} while deferring all measurements to the end, as is customary when analysing quantum algorithms.

Theorem 3. *Let SS be a homomorphic, linear, and well-formed (m, m', t) secure sketch over \mathcal{M} . Then HFE is a $(\mathcal{M}, m, \ell, t, \epsilon = 2q \cdot 2^{-m'/2}, \epsilon_1 = 2q \cdot 2^{-m'/2}, \epsilon_2)$ reusable robust fuzzy extractor with reusable robustness*

$$\epsilon_2 \leq \min \left\{ \begin{array}{l} q \cdot 2^{-m'} + 2\sqrt{q \cdot 2^{-m'}} \cdot \left(\sqrt{\frac{1}{2^k} + q + 1} \right) + \frac{2}{2^k}, \\ \left(\sqrt{q \cdot 2^{-m'}} + 2\sqrt{1/2^k} + 2\sqrt{q(q+1) \cdot 2^{-m'}} \right)^2 \end{array} \right\}$$

in the QROM with respect to unbounded quantum algorithms making at most q quantum random oracle queries.

Proof. For the expressions of ϵ and ϵ_1 , see Lemma 3 and its proof in Sect. 4.1. We now restrict attention to the bound on ϵ_2 . Assume that \mathcal{A} takes its deferred measurement form (see above the theorem statement and further details in Sect. D). Consider the algorithm $\mathcal{E}_{\mathcal{A}}^H$ that essentially represents the reusable robustness experiment between \mathcal{A} and its challenger. Specifically, for any random oracle H parsed as H_1, H_2 (where H_1 consists of the first k output bits and H_2 the remaining ℓ bits), the details of $\mathcal{E}_{\mathcal{A}}^H$ are as follows:

- The input of \mathcal{E}_A^H takes the form $z = (w, z' := (s, (h_\Delta)_{\|\Delta\| \leq t}))$, where $w \leftarrow \mathcal{W}$, $s := \text{SS.Gen}(w)$, $s_\Delta = \text{SS.Gen}(w + \Delta) = s + \text{SS.Gen}(\Delta)$ and $h_\Delta := H(w + \Delta, s_\Delta)$ is parsed as $(h_\Delta^{(1)}, h_\Delta^{(2)}) \in \{0, 1\}^k \times \{0, 1\}^\ell$.
- $\mathcal{E}_A^H(z)$ runs $\mathcal{A}^{H, \mathcal{O}_{\text{Gen}}}$ using the copying strategy to defer all \mathcal{O}_{Gen} query input measurements to the end by keeping a database D_{Gen} . Recall that this requires *quantum* access to \mathcal{O}_{Gen} . In order to achieve this, \mathcal{E}_A^H simply uses the values $z' = (s, (h_\Delta)_{\|\Delta\| \leq t})$ to implement the *quantum* oracle \mathcal{O}_{Gen} for \mathcal{A} .
- $\mathcal{E}_A^H(z)$ measures the D_{Gen} registers and then \mathcal{A} 's output registers to obtain results D_{Gen}^* and output $((s^*, h^*), \Delta^*)$.
- $\mathcal{E}_A^H(z)$ then decides its final output by performing the following:
 - If $\|\Delta^*\| > t$, output 0.
 - If there is a $\Delta \in D_{\text{Gen}}^*$ such that $(s_\Delta, h_\Delta) = (s^*, h^*)$, output 0.
 - If there is a Δ such that $s^* = s_\Delta$, $\|\Delta\| \leq t$ and $\|\Delta^* - \Delta\| \leq t$, then check whether $h^* = h_\Delta^{(1)}$. If so, output 1 and if not output 0.
 - Otherwise, use a random oracle query to check whether the random oracle evaluated at $(w + f(\Delta^*, s^*, s), s^*)$ has its first k bits equal to h^* , where f is the function from the linearity property. If so, output 1 and if not output 0.

Throughout this proof, we will abuse notation by writing $\mathcal{A}^H(z')$ to represent running $\mathcal{A}^{H, \mathcal{O}_{\text{Gen}}}$, where \mathcal{O}_{Gen} queries are answered using oracle access to the appropriate entries in z' . Using this notation, the second item in the description of \mathcal{E}_A^H simply says that \mathcal{E}_A^H runs $\mathcal{A}^H(z')$. Also, the conditions that \mathcal{E}_A^H checks are identical to those checked in the reusable robustness experiment with respect to the construction of HFE. The only slight difference is that the values $h_\Delta^{(1)}$ are used to verify whether $H(\text{SS.Rec}(w + \Delta^*, s^*), s^*) = h^*$ in the case where $s^* = s_\Delta$ for $\|\Delta^* - \Delta\| \leq t$, $\|\Delta\| \leq t$. Note that in this case, $\text{SS.Rec}(w + \Delta^*, s^*) = w + \Delta$, meaning that the value $h_\Delta^{(1)}$ may be used to correctly perform the check instead of using the random oracle directly. Therefore, we have that $\epsilon_2 = \Pr[\mathcal{E}_A^H(z) = 1]$. We now focus on bounding $\Pr[\mathcal{E}_A^H(z) = 1]$ to complete the proof.

We will be applying the O2H lemma (Lemma 1) using the notation introduced above. To do so, consider sampling a uniformly random function G , $w \leftarrow \mathcal{W}$ and setting H to be equal to G apart from on the set $S := \{(w + \Delta, s_\Delta) : \|\Delta\| \leq t\}$. On the set S , H is sampled as a uniformly random function. Note that H is (on its own) a uniformly random function as required. Intuitively, if \mathcal{A} has access to G , the reusability oracle (which uses values of H evaluated on points in S) does not help it identify w . Lemma 1 then tells us that

$$|\Pr[\mathcal{E}_A^H(z) = 1] - \Pr[\mathcal{E}_A^G(z) = 1]| \leq 2(q+1)\sqrt{\text{Adv}_B} \quad (1)$$

$$\left| \sqrt{\Pr[\mathcal{E}_A^H(z) = 1]} - \sqrt{\Pr[\mathcal{E}_A^G(z) = 1]} \right| \leq 2(q+1)\sqrt{\text{Adv}_B} \quad (2)$$

where \mathcal{B} is described in Lemma 1 and $\text{Adv}_B := \Pr[\mathcal{B}^G(z) \in S]$. The factor $(q+1)$ on the right-hand side arises because \mathcal{E}_A^G makes at most one extra query to the random oracle when deciding whether to output 1 or 0. We can now bound $\Pr[\mathcal{E}_A^G(z) = 1]$ and Adv_B using the two lemmas below to complete the proof.

Lemma 4. $\Pr[\mathcal{E}_A^G(z) = 1] \leq q \cdot 2^{-m'} + 2\sqrt{\frac{q2^{-m'}}{2^k}} + \frac{2}{2^k}$ and

$$\sqrt{\Pr[\mathcal{E}_A^G(z) = 1]} \leq \sqrt{q \cdot 2^{-m'}} + 2\sqrt{\frac{1}{2^k}}.$$

Proof (Of lemma). Consider the algorithm $\bar{\mathcal{E}}_A^G$ that on input z (sampled like the input of \mathcal{E}_A^G) behaves as follows:

- Run \mathcal{E}_A^G up to the point where the output registers of \mathcal{A} and the D_{Gen} register have just been measured to obtain $(\Delta^*, s^*, h^*, D_{\text{Gen}}^*)$.
- Classically compute the output

$$(x := (w + f(\Delta^*, s^*, s), s^*), y := h^*, \zeta := (\Delta^*, s^*, D_{\text{Gen}}^*)).$$

Now define the set $\mathcal{R} := \{(x, y, (\Delta^*, s^*, D_{\text{Gen}})) : \|\Delta^*\| \leq t, s^* \neq s_\Delta \forall \Delta \in D_{\text{Gen}}\}$. Note that there are two ways in which \mathcal{E}_A^G checks the hash value h^* when deciding whether to output 1 or 0: in the case that $s^* = s_\Delta$ for some $\|\Delta\| \leq t$ and $\|\Delta^* - \Delta\| \leq t$, it uses $h_\Delta^{(1)}$ and otherwise it uses the random oracle to perform the check. In this first case, \mathcal{E}_A^G outputs 1 only if $h^* = h_\Delta^{(1)}$ and $\Delta \notin D_{\text{Gen}}$. Therefore, given the first case, the probability that \mathcal{E}_A^G outputs 1 is at most 2^{-k} because h_Δ is a uniform value, independent of G , that \mathcal{A} never accesses. To argue this formally, recall that our deferred measurement view where \mathcal{A} uses quantum access to \mathcal{O}_{Gen} is equivalent to the original view where \mathcal{A} takes intermediate measurements before accessing \mathcal{O}_{Gen} classically. In the second case, \mathcal{E}_A^G outputs 1 when the output of $\bar{\mathcal{E}}_A^G$ denoted as (x, y, ζ) is in \mathcal{R} and $G_1(x) = y$ where G_1 is the function that outputs the first k output bits of G . Denoting $p := \Pr[\bar{\mathcal{E}}_A^G(z) \in \mathcal{R} \wedge G_1(x) = y]$, we then have that $\Pr[\mathcal{E}_A^G(z) = 1] \leq p + 1/2^k$ after considering both of the aforementioned cases. We can also use the triangle inequality to obtain $\sqrt{\Pr[\mathcal{E}_A^G(z) = 1]} \leq \sqrt{p} + \sqrt{1/2^k}$.

Now define G_2 to be function that outputs the final ℓ bits of G so that G is parsed as a concatenation G_1 and G_2 . Next, consider running algorithm $\bar{\mathcal{E}}_A^{\text{CStO}, G_2}(z)$ where G_1 is implemented using CStO from Lemma 2 and G_2 is implemented using the standard unitary. Let p' denote the probability that on running $\bar{\mathcal{E}}_A^{\text{CStO}, G_2}(z)$, we get output $(x, y, \zeta) \in \mathcal{R}$, and that on measuring the CStO database D , we find that $D(x) = y \neq \perp$. By Lemma 2, we know that $\sqrt{p} \leq \sqrt{p'} + \sqrt{1/2^k}$ where k is the output length of G_1 .

To complete the proof of this lemma we bound p' . Note that if $D(x) \neq \perp$ and $(x, y, \zeta) \in \mathcal{R}$, we are guaranteed that $(w + f(\Delta^*, s^*, s), s^*)$ is present in some element of the database D . Therefore, p' is at most the probability that $\mathcal{A}^{\text{CStO}, G_2}(z')$ simply ends up inserting $(w + f(\Delta^*, s^*, s), s^*)$ into the database. Denote the probability of this latter event as p'' . Since $p' \leq p''$ we now attempt to bound p'' . Consider the following algorithm $\bar{\mathcal{A}}$ that takes as input s (where $s = \text{SS.Gen}(w)$ for $w \leftarrow \mathcal{W}$) and attempts to recover w :

- Sample h_Δ uniformly for each Δ with $\|\Delta\| \leq t$ and define $z' := (s, (h_\Delta)_\Delta)$.

- Run $\mathcal{A}^{\text{CStO}, G_2}(z')$ to obtain output (Δ^*, s^*, h^*) and measured \mathcal{O}_{Gen} query list D_{Gen}^* , then subsequently measure the CStO database to obtain

$$D = \{(x_1, y_1), \dots, (x_{q'}, y_{q'})\} \text{ where } q' \leq q.$$

- Sample $i \leftarrow [q']$, parse $x_i = (w_i, s_i)$ and output $w_i - f(\Delta^*, s^*, s)$.

The input z' of $\mathcal{A}^{\text{CStO}, G_2}$ here is clearly from the correct distribution as it is s along with a collection of random values independent of the random oracle (CStO, G_2). Note that $\bar{\mathcal{A}}(s)$ returns w if and only if $\mathcal{A}^{\text{CStO}, G_2}(z')$ inserts $(w + f(\Delta^*, s^*, s), \star)$ into D and the index i corresponds to a position of an element of the form $(w + f(\Delta^*, s^*, s), \star)$ in D . Therefore, we have $\Pr[\bar{\mathcal{A}}(s) = w] \geq p''/q'$ where we have inequality because there may be more than one element of the desired form. We also know that information theoretically, $\Pr[\bar{\mathcal{A}}(s) = w] \leq 2^{-m'}$. Overall, we may conclude that $p' \leq p'' \leq q' \cdot 2^{-m'} \leq q \cdot 2^{-m'}$. Plugging this into the inequality $\sqrt{p} \leq \sqrt{p'} + \sqrt{1/2^k}$ and $\Pr[\mathcal{E}_{\mathcal{A}}^G(z) = 1] \leq p + 1/2^k$ or $\sqrt{\Pr[\mathcal{E}_{\mathcal{A}}^G(z) = 1]} \leq \sqrt{p} + \sqrt{1/2^k}$ completes the proof of this lemma. \square

Lemma 5. *Assuming that SS is a well-formed, homomorphic (m, m', t) secure sketch, $\text{Adv}_{\mathcal{B}} \leq \frac{q}{q+1} \cdot 2^{-m'}$.*

Proof (Of lemma). Recall that all queries to the random oracle during execution of $\mathcal{E}_{\mathcal{A}}^G$ (except potentially the last one) are performed when running \mathcal{A} as a subroutine. Also, recall that when deciding whether to output 1 or 0, $\mathcal{E}_{\mathcal{A}}^G$ only ever queries a point of the form (\star, s_{Δ}) to the random oracle when $s^* = s_{\Delta}$ and $\|\Delta^* - \Delta\| > t$. In such a case, we have that $\text{SS.Rec}(w + \Delta^*, s_{\Delta}) \neq w + \Delta$ as SS is a well-formed sketch and $\|(w + \Delta^*) - (w + \Delta)\| > t$. This implies that if \mathcal{B}^G chooses to measure the input register just before the random oracle query used to check h^* , it cannot possibly obtain a value in $S := \{(w + \Delta, s_{\Delta}) : \|\Delta\| \leq t\}$.

Now we consider the case that the query randomly chosen by \mathcal{B}^G does not correspond to the final query. In this case, all queries are the result of running $\mathcal{A}^G(z')$ where $z' = (s, (h_{\Delta})_{\|\Delta\| \leq t})$ is just s along with a collection of uniform values independent of the random oracle G . Therefore, if \mathcal{B}^G manages to output an element of S with some probability, then there is an unbounded quantum algorithm taking input s that can output an element of S with the same probability. This (unbounded) quantum algorithm just uses s , chooses uniform G and values $(h_{\Delta})_{\|\Delta\| \leq t}$ and then runs $\mathcal{A}^G(z')$ where $z' := (s, (h_{\Delta})_{\|\Delta\| \leq t})$, measuring a random query. Now, from any element (w', s_{Δ}) of S and knowledge of s , we can compute $\Delta = \text{SS.Rec}(0, s_{\Delta} - s)$ and $w = w' - \Delta$. The correctness of this procedure is due to the fact that for a homomorphic sketch capable of correcting t errors, $\text{SS.Rec}(0, \text{SS.Gen}(\Delta)) = \Delta$. Using this procedure, we can compute w from any element of S and s . Therefore, in the event that the randomly chosen query of \mathcal{B}^G does not correspond to the last query, if \mathcal{B}^G returns an element of S with probability \tilde{p} , then there is a quantum algorithm that returns w with probability \tilde{p} . However, we know that information theoretically, $\tilde{p} \leq 2^{-m'}$.

To complete the proof of the lemma, note that the randomly chosen query that \mathcal{B} measures is equal to the final query with probability at least $1/(q+1)$. \square

Inserting the lemma bounds into Eqs. (1) and (2) completes the proof of reusable robustness. \square

Remark 1. We may drop the well-formedness of SS in favour of a weaker property: $\forall w$ and $t < \|\Delta\| \leq 2t$, $\text{SS.Rec}(w + \Delta, \text{SS.Gen}(w)) \neq w$. This allows to argue (as in the above proof) \mathcal{E}_A^G never queries the random oracle on points in S when checking h^* . Alternatively, one can use a $(m, m', 2t)$ secure sketch, changing the third output condition of \mathcal{E}_A^H to consider the case where $s^* = s_\Delta$ for any $\Delta \leq 2t$.

4.3 Parameters and Efficiency

As mentioned at the beginning of this section, we require a $(m, m' \approx 4\lambda, t)$ linear, homomorphic and well-formed secure sketch for λ -bit security. This is independent of the length of the key ℓ we derive from HFE per hint value. To use the syndrome code-offset secure sketch (see Sect. 2.2), we need a linear $[\eta, \kappa, d = 2t + 1]$ code and a source distribution \mathcal{W} with support over $\{0, 1\}^\eta$ and min-entropy $m = m' + (\eta - \kappa) \approx 4\lambda + (\eta - \kappa)$. For an almost perfect source, $m \approx \eta$ which means choosing a linear $[\eta, \kappa, 2t + 1]$ code with $\kappa \geq 4\lambda$. This may introduce a rather stringent requirement on η and m (which corresponds to the number of source bits and entropy required) for reasonable error rates. In particular, as t increases, the size of the term $\eta - \kappa$ must increase, meaning that the required entropy of the source m increases too. However, in all other aspects, the construction is very efficient: only $\eta - \kappa + \lambda$ bits of storage are required per hint and we can extract *any number* of random bits ℓ using the random oracle. In the classical ROM setting, to correct t errors, we can simply use a $[\eta, \kappa, d = 2t + 1]$ error correcting code where $\kappa \geq 2\lambda$ for λ -bit security.

5 Post-Quantum Standard Model Construction

In this section, we present an improvement to the standard model construction of Cui et al. [12] based on the post-quantum assumption learning parity with noise (LPN). The scheme is built from three components: a linear homomorphic secure sketch (e.g. the syndrome code-offset construction), a randomness extractor, and a symmetric key encapsulation mechanism (SKEM) that has a decapsulation uniformity property. The notions of randomness extractors and SKEMs are defined in Appendix B whereas the (ring-)LPN problem is recalled in Appendix C. Figure 1 depicts how the components fit together to make the reusable robust fuzzy extractor. Note that this construction is in the CRS model where the CRS takes the form of a uniform seed used in a randomness extractor. We recall the LPN-based SKEM_{LPN} from [12]:

- $\text{SKEM}_{LPN}.\text{Setup}(1^\lambda)$: Derive public parameters $m = \text{poly}(\lambda), n = \text{poly}(\lambda), k = \text{poly}(\lambda), \tau \in (0, 1/2)$ and pick a $[m, k, d = 2t + 1]$ code $(\mathcal{E}, \mathcal{D})$ s.t. $\tau m \leq t$.
- $\text{SKEM}_{LPN}.\text{Encaps}(\mathbf{K})$: For $\mathbf{K} \in \mathbb{Z}_2^{n \times k}$, sample $\mathbf{a} \in \mathbb{Z}_2^n \setminus \{0\}, \mathbf{e} \leftarrow \text{Ber}_\eta^m, \mathbf{k}' \leftarrow \mathbb{Z}_2^k$, output $((\mathbf{c}_1, \mathbf{c}_2) := (\mathbf{a}, \mathbf{a}^\top \mathbf{K} + \mathbf{e}^\top + \mathcal{E}(\mathbf{k}')), \mathbf{k}')$.
- $\text{SKEM}_{LPN}.\text{Decaps}((\mathbf{c}_1, \mathbf{c}_2), \mathbf{K})$: Output $\mathcal{D}(\mathbf{c}_2 - \mathbf{c}_1 \mathbf{K})$.

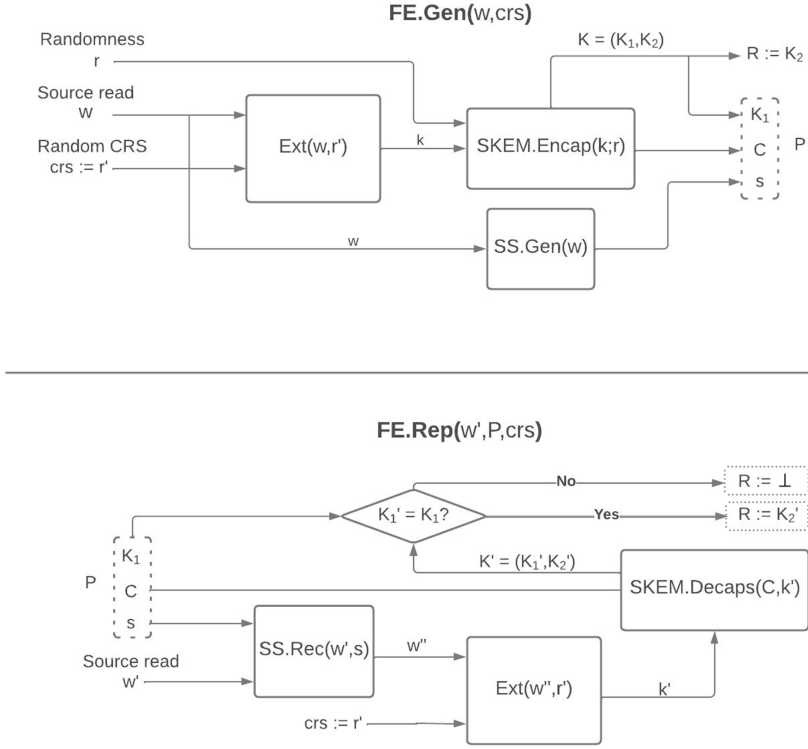


Fig. 1. A diagram of the robustly reusable fuzzy extractor of [12]. SS is a linear, homomorphic secure sketch, Ext is a strong randomness extractor and SKEM is a SKEM with decapsulation uniformity. An algorithm FE.Init samples a random seed (crs) for the extractor. The “authentication tag” K_1 must be $\omega(\log(\lambda))$ bits, with K_2 the remaining bits of K . K_2 is the fuzzy extractor output and P is the helper data.

Assuming k and n are $O(\lambda)$, the secret key in the above SKEM has $O(\lambda^2)$ bits. Given Fig. 1, we need to extract $O(\lambda^2)$ uniform bits from a randomness extractor to use this SKEM. In the PUF setting, $O(\lambda^2)$ source bits would be required. We next propose a ring-LPN based SKEM (with decapsulation uniformity) to minimize the number of secret key bits (and therefore source bits) below. The description interprets ring elements as coefficient vectors when applying linear codes. Assuming k and n are $O(\lambda)$, we end up with encapsulation keys of size $O(\lambda)$ which is an improvement by a factor of λ . Explicitly, our proposal is:

- $\text{SKEM}_{RLPN}.\text{Setup}(\lambda)$: Choose a degree $n = \text{poly}(\lambda)$ polynomial f , set $R := \mathbb{Z}[X]/f(X)$ and $R_2 := R/2R$. Choose k and $m = \text{poly}(\lambda)$ s.t. $m = \kappa n$ for some $\kappa \in \mathbb{Z}$, $\tau \in (0, 1/2)$ and pick a $[m, k, d = 2t + 1]$ code $(\mathcal{E}, \mathcal{D})$ s.t. $\tau m \leq t$.
- $\text{SKEM}_{RLPN}.\text{Encaps}((\mathbf{k}_1, k_2))$: For $\mathbf{k}_1 \in R_2^\kappa$, $k_2 \in \mathbb{Z}_2^k$, sample $a \in R_2 \setminus \{0\}$, $\mathbf{e} \leftarrow \text{Ber}_{\tau}^{\kappa \cdot n}$ (interpreted as $\mathbf{e} \in R_2^\kappa$), $\tilde{k} \leftarrow \mathbb{Z}_2^k$ and output

$$(c_1, c_2) := (a, a \cdot \mathbf{k}_1 + \mathbf{e} + \mathcal{E}(\tilde{k})), \quad k' = \tilde{k} \oplus k_2.$$

– $\text{SKEM}_{RLPN}.\text{Decaps}((c_1, c_2), (\mathbf{k}_1, k_2))$: Output $\mathcal{D}(c_2 - c_1 \mathbf{k}_1) \oplus k_2$.

Remark 2. The main difference is the “one-time pad” k_2 which is a uniform hidden value from the perspective of the decapsulation uniformity definition (see Definition 9 in Appendix B). If R_2 is a field, $c_1 \neq 0$ and $\mathbf{k}_1 \in R_2^n$ is uniform, $c_1 \cdot \mathbf{k}$ is uniform without k_2 . If R_2 is not a field, entries of $c_1 \cdot \mathbf{k}$ all lie in the ideal generated by c_1 , so k_2 ensures decapsulation uniformity. Choosing a non-field R_2 can lead to efficient implementations of ring multiplication e.g. [2].

The proof of correctness and key-shift pseudorandomness are very similar to the scheme of Cui et al. and are deferred to Appendix F. We now ask how many source bits/entropy is required in the robustly reusable fuzzy extractor of Cui et al. Suppose we have \bar{n} source bits with \bar{m} bits of min-entropy. Then the entropy left after revealing a syndrome code-offset secure sketch using a $[\bar{n}, \bar{k}, \bar{d} = 2\bar{t} + 1]$ code is $\bar{m} - (\bar{n} - \bar{k})$ bits. After applying a randomness extractor (e.g. the Toeplitz extractor in Appendix B) we derive at most $\bar{m} - \bar{n} + \bar{k} - 2\lambda$ uniform bits (within statistical distance $2^{-\lambda}$).

We will now use the fact that the Cui et al. construction uses extracted random bits as a SKEM key. In the case that $\kappa = 1$ in SKEM_{RLPN} and R_2 is a field (see the above remark arguing that we can then ignore k_2), we require that $\bar{m} - \bar{n} + \bar{k} \geq 2\lambda + n$ where n is the dimension of a ring R chosen so that $\text{RLPN}_{R,1,\tau}$ has λ bits of security. For plausible parameters, n is much larger than 2λ so the term $2\lambda + n$ will be much larger than the analogous term of 4λ (see Sect. 4.3) for the random oracle based HFE construction. To be more concrete, the updated ring-LPN Lepton parameters³, suggests that n is at least 2^{15} at the 128-bit security level, leading to a very rough estimation that $n = 256\lambda$ for practical ring-LPN schemes. Therefore, not only does the standard model construction introduce *much stricter* requirements on the number of required source bits/entropy, it also requires the storage of much larger hints compared to the HFE construction. In addition, the standard model construction requires access to a true random number generator unlike the HFE construction. We stress here that the security of the HFE construction only holds in an *idealized* quantum random oracle model, so comparisons with the ring-LPN construction needs to be considered carefully.

6 Conclusion

In conclusion, we have given a proof of security for a robustly reusable fuzzy extractor in the QROM. Our construction is simple and the security bounds are very concrete, allowing a relatively easy application of our results to practical situations. Prior to our work, we are not aware of any fuzzy extractor results in the QROM. Therefore, our contributions may serve as a baseline for future fuzzy extractor constructions in the QROM. We also suggested an optimisation to the

³ Available in the penultimate presentation slide of <https://csrc.nist.gov/CSRC/media/Presentations/Lepton/images-media/Lepton-April2018.pdf>.

only existing post-quantum robustly reusable fuzzy extractor in the standard model capable of correcting a linear fraction of errors (as required in the PUF setting), showing that the QROM construction outperforms it by a distance.

Finally, we suggest directions for future work. Recall that there is a model slightly stronger than the one considered in this work where errors in the security arguments may depend arbitrarily on secret values, rather than just values that the adversary sees [8, 9]. The first potential avenue for future work is to come up with efficient constructions in this stronger model, where security bounds depend weakly on the number of correctable errors and a linear fraction of errors can be tolerated. Another direction is to improve the security bounds in our QROM proofs.

Acknowledgements. We thank Prof. Kenneth G. Paterson, Dr. Shahram Mossayebi and our anonymous reviewers for their comments and suggestions on this work.

A Quantum Computation Preliminaries

Let \mathcal{H} be a complex Hilbert space of dimension n with orthonormal basis

$$\{|1\rangle, \dots, |n\rangle\}.$$

A (pure) quantum state $|\psi\rangle$ over this Hilbert space is a (normalized) complex linear combination of the basis states i.e. $|\psi\rangle = \sum_{i=1}^n \alpha_i |i\rangle$ where $\alpha_i \in \mathbb{C}$ and $\sum_i \|\alpha_i\|^2 = 1$. On making a basis measurement on the state $|\psi\rangle$, we have $\Pr[i] = \|\alpha_i\|^2$ where $\Pr[i]$ is the probability of measuring the state to be in $|i\rangle$. More generally, the probability of measuring the state $|\psi\rangle$ and finding it to be in some other state $|\phi\rangle$ is $\|\langle\psi|\phi\rangle\|^2$ which is equal to $\text{Tr}(|\phi\rangle\langle\phi| |\psi\rangle\langle\psi|) = \text{Tr}(\Pi_\phi |\psi\rangle\langle\psi|)$. Next, a projection value measurement (PVM) with k possible outcomes is a collection of k projections as Π_1, \dots, Π_k such that $\sum_{i=1}^k \Pi_i = I$. Then the probability of the outcome indexed by i is $\Pr[i] = \langle\psi|\Pi_i|\psi\rangle$. The post-measurement state after measuring i is $\Pi_i |\psi\rangle / \|\Pi_i |\psi\rangle\|$.

Quantum algorithms are usually described by applying some quantum computation i.e. unitary U to a starting state $|\varphi_0\rangle$. The state $|\varphi_0\rangle$ is said to contain input registers, output registers and ancillary registers to aide its computation. At the end of this computation, a measurement associated with some set of projections projection $\{\Pi_1, \dots, \Pi_k\}$ over the output registers is taken to obtain some usable classical information from the computation. In this context, k will be the number of distinct values the output registers can take. For example, if the output registers consist of κ qubit registers (i.e. registers that can take only binary values when measured), then $k = 2^\kappa$. Overall, the probability of the algorithm outputting j is $\|\Pi_j \cdot U |\varphi_0\rangle\|^2$ where U denotes the unitary applied when running the quantum algorithm. Note that this format of a quantum algorithm is w.l.o.g as any algorithm with intermediate projections can be written in the stated form (with slightly different unitaries and projections). In other words, we do not need to consider intermediate measurements by the *principle of deferred measurements*.

B Preliminaries for Standard Model Construction

The following definition uses the notion of statistical distance. The statistical distance of two distributions P and Q is $SD(P, Q) := \sum_x |\Pr[P = x] - \Pr[Q = x]|/2$.

Definition 8. A function $\text{Ext} : W \times S \rightarrow R$ is a strong (m, ϵ) randomness extractor if for any distribution \mathcal{W} over W with $\tilde{H}_\infty(\mathcal{W}) \geq m$,

$$SD((\text{Ext}(\mathcal{W}, U_S), U_S), U_{R \times S}) \leq \epsilon$$

where U_S and $U_{R \times S}$ are the uniform distributions over S and $R \times S$ respectively.

Example 2. Take S to be the set of all binary Toeplitz matrices of the form

$$\mathbf{A} = \begin{bmatrix} a_0 & a_{n-1} & \dots & a_1 \\ a_1 & a_0 & \dots & a_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{k-1} & a_{k-2} & \dots & a_k \end{bmatrix} \in \mathbb{Z}_2^{k \times n}$$

where $k < n$ and $W = \mathbb{Z}_2^n$. Then $\text{Ext}(\mathbf{w}, \mathbf{A}) = \mathbf{A} \cdot \mathbf{w}$ is a strong $(m, 2^{-(m-k)/2})$ extractor. Informally, this extractor maps m bits of entropy to within statistical distance $2^{-\lambda}$ of the uniform distribution over \mathbb{Z}_2^k provided that $m \geq k + 2\lambda$.

Definition 9 ([12, 26]). A symmetric key encapsulation mechanism (SKEM) with decapsulation uniformity consists of three algorithms

$$\text{SKEM} = (\text{SKEM.Setup}, \text{SKEM.Encaps}, \text{SKEM.Decaps})$$

where:

- $\text{SKEM.Setup}(1^\lambda)$ takes as input a security parameter and outputs public parameters pp that include descriptions of spaces $\mathcal{R}_e, \mathcal{K}, \mathcal{K}'$ and \mathcal{C} .
- $\text{SKEM.Encaps}(k; r)$ is a probabilistic algorithm taking as input a key $k \in \mathcal{K}'$ and randomness $r \in \mathcal{R}_e$ and outputs (C, K) where $C \in \mathcal{C}$ is a ciphertext and $K \in \mathcal{K}$ is an encapsulated key.
- $\text{SKEM.Decaps}(C, k)$ takes as input a key $k \in \mathcal{K}'$ and ciphertext $C \in \mathcal{C}$ and outputs a decrypted key $K' \in \mathcal{K}$ or \perp .

A SKEM with decapsulation uniformity must satisfy the following properties:

1. (Correctness)

$$\Pr \left[K = K' : \begin{array}{l} \text{pp} \leftarrow \text{SKEM.Setup}(\lambda), k \leftarrow \mathcal{K}' \\ (C, K) \leftarrow \text{SKEM.Encaps}(k) \\ K' \leftarrow \text{SKEM.Decaps}(C, k) \end{array} \right] = 1 - \text{negl}(\lambda).$$

2. (Key-Shift Pseudorandomness) For all PPT \mathcal{A} ,

$$\Pr \left[b' = b : \begin{array}{l} \text{pp} \leftarrow \text{SKEM.Setup}(\lambda), k \leftarrow \mathcal{K}' \\ b \leftarrow \{0, 1\} \\ b' \leftarrow \mathcal{A}_b^{\text{O}_b^{\text{ksp}}}(\lambda) \end{array} \right] = \text{negl}(\lambda)$$

where the oracles O_0^{ksp} and O_1^{ksp} are defined below.

3. (*Decapsulation Uniformity*) For any $C \in \mathcal{C}$ and $K' \in \mathcal{K}$,

$$\Pr[\text{SKEM.Decaps}(C, k) = K' : \text{pp} \leftarrow \text{SKEM.Setup}(\lambda) \underset{k \leftarrow \mathcal{K}'}{\text{pp}}] = \frac{1}{|\mathcal{K}|}.$$

The oracles for the key-shift pseudorandomness property are defined as:

- $\mathcal{O}_0^{\text{ksp}}$: On input δ , return uniform $(C, K) \leftarrow \mathcal{C} \times \mathcal{K}$.
- $\mathcal{O}_1^{\text{ksp}}$: On input δ , return $(C, K) \leftarrow \text{SKEM.Encaps}(k + \delta)$.

C (Ring-)LPN Preliminaries

We now recall the LWE [22] and ring-LWE [20] problems with modulus 2, also known as LPN and ring-LPN (RLPN) respectively. Below, Ber_τ represents the Bernoulli distribution with probability $\tau \in [0, 1]$.

Definition 10. For dimensions m, n and Bernoulli parameter $\tau \in (0, 1/2)$, the LPN distribution with secret $\mathbf{S} \in \mathbb{Z}_2^{n \times m}$, denoted $A_{m,n,\tau}(\mathbf{S})$ is sampled as follows: First sample $\mathbf{a} \leftarrow \mathbb{Z}_2^n$, $\mathbf{e} \leftarrow \text{Ber}_\tau^m$ and output $(\mathbf{a}, \mathbf{a}^\top \mathbf{S} \oplus \mathbf{e})$. The $\text{LPN}_{m,n,\tau}$ problem is to distinguish between an unbounded number of samples of $A_{m,n,\chi}(\mathbf{S})$ and the uniform distribution over $\mathbb{Z}_2^n \times \mathbb{Z}_q^m$ where $\mathbf{S} \leftarrow \mathbb{Z}_2^{n \times m}$.

Definition 11 ([17]). For dimension m' , ring $R = \mathbb{Z}[X]/(f(X))$ with f of degree n , and Bernoulli parameter $\tau \in (0, 1/2)$, define $R_2 := R/2R$. The ring-LPN distribution with secret $\mathbf{s} \in R_q^{m'}$ denoted as $A_{m',R,\tau}(\mathbf{s})$ is sampled as follows: First sample $a \leftarrow R_2$, $\mathbf{e} \leftarrow \text{Ber}_\tau^{m'n}$ (interpreted as an element of $R_2^{m'}$ where each coefficient is sampled from Ber_τ) and then output $(a, a \cdot \mathbf{s} + \mathbf{e}) \in R_2 \times R_2^{m'}$. The $\text{RLPN}_{m',R,\tau}$ problem is to distinguish between an unbounded number of samples of $A_{m',R,\tau}(\mathbf{s})$ and the uniform distribution over $R_2 \times R_2^{m'}$ where $\mathbf{s} \leftarrow R_2^{m'}$.

Note that a single secret i.e. $m' = 1$ RLPN sample is essentially $(a, a \cdot s + e)$, where $a \cdot s$ can be expressed as a $n \times n$ matrix multiplied by a n -dimensional vector. In other words, a single RLPN sample $(a, a \cdot s + e)$ can be considered as n structured LPN samples (with $m = 1$). However, only $2n$ bits are required to represent an RLPN sample whereas n LPN samples require $n^2 + n$ bits of storage. Further, ring multiplication can be made extremely fast if the underlying ring splits into many factors by applying fast Fourier transform-like techniques (e.g. [2]). These are two main advantages of RLPN over LPN.

D Deferring Measurements for Recorded Classical Oracle Queries

Consider a quantum algorithm \mathcal{A} that is restricted to making *classical* queries to some *quantum* instantiation of an oracle $\mathcal{O} : X \rightarrow Y$. This is no different from the case where \mathcal{A} has purely classical access to the oracle. In order to make a classical query to the quantum oracle, \mathcal{A} can take an intermediate measurements

of the query input register before making its query. Note that this is equivalent to a classical query because the query input register collapses to a classical state after the measurement. Our aim here is to show that there is an algorithm $\bar{\mathcal{A}}$ that keeps some database registers (in addition to the registers of \mathcal{A}) such that

- $\bar{\mathcal{A}}$ does not make any measurements until it is ready to make an output measurement (in particular, there are no intermediate measurements of the query input register).
- When producing output, $\bar{\mathcal{A}}$ additionally measures the database registers.
- The joint distribution of \mathcal{A} 's output and intermediate measurements is identical to the joint distribution of $\bar{\mathcal{A}}$'s output and database measurement.

Given these properties, $\bar{\mathcal{A}}$ perfectly simulates the behaviour of \mathcal{A} and its intermediate measurements while deferring measurements to the end of the computation.

Note that we can describe the measurement of the query input register by considering the set of projections $\{\Pi_x = \mathbb{I} \otimes |x\rangle\langle *|x \otimes \mathbb{I} : x \in X\}$. We explicitly describe the algorithm \mathcal{A} making q classical queries to \mathcal{O} as follows:

- \mathcal{A} has registers $|*\rangle \dots_{\mathcal{A}} \otimes |*\rangle x, y_{\mathcal{O}}$ where for any basis state, applying the oracle treats x as the input register and y as the output, i.e. an oracle application is

$$(\mathbb{I} \otimes \mathcal{O}) |\phi\rangle_{\mathcal{A}} \otimes |x, y\rangle_{\mathcal{O}} := |\phi\rangle_{\mathcal{A}} \otimes |x, y \oplus \mathcal{O}(x)\rangle_{\mathcal{O}}.$$

- \mathcal{A} begins in the state $|\phi_0\rangle$.
- For $i = 1, \dots, q$:
 - \mathcal{A} performs a unitary U_i on the entire state to obtain $|\phi_{i-1}\rangle$.
 - \mathcal{A} performs a measurement. Denoting the result as x_i , the un-normalized collapsed state is $(\mathbb{I} \otimes \Pi_{x_i} \otimes \mathbb{I}) |\phi_{i-1}\rangle$.
 - \mathcal{A} sends its state to the oracle and it applies $\mathbb{I} \otimes \mathcal{O}$.
- \mathcal{A} applies a unitary U_{q+1} and takes a measurement of its output registers.

Note that if we were to add a register containing the results x_i after every measurement, the output distribution of \mathcal{A} is unchanged as the overall state is always a product state between \mathcal{A} 's state and a classical list of observed measurements.

Here we will show how to defer measurements of \mathcal{A} to the end while preserving the output distribution of \mathcal{A} and the intermediate measurements jointly. There will be three sets of registers of the form $|\dots\rangle_{\mathcal{A}}$, $|x, y\rangle_{\mathcal{O}}$ and $|\dots\rangle_D$ where the last of these denotes a set of q registers that will store a record of the \mathcal{O} -queries. The deferred measurement algorithm will be denoted as $\bar{\mathcal{A}}$ and will have *quantum* access to \mathcal{O} . The behaviour of $\bar{\mathcal{A}}$ is as follows:

- $\bar{\mathcal{A}}$ has registers $|\dots\rangle_{\mathcal{A}} \otimes |x, y\rangle_{\mathcal{O}} \otimes \underbrace{|\dots\rangle_D}_{q \text{ times}}$.
- $\bar{\mathcal{A}}$ begins in the state $|\phi_0\rangle \otimes \overbrace{|0, \dots, 0\rangle_D}^q$ where $|\phi_0\rangle$ is \mathcal{A} 's starting state.
- For $i = 1, \dots, q$:
 - $\bar{\mathcal{A}}$ performs the unitary $U_i \otimes \mathbb{I}_D$ (where U_i is the same unitary as in the description of \mathcal{A}).

- $\bar{\mathcal{A}}$ then adds the contents of the query input register into the i -th slot of the D registers. Denote the unitary that performs this as C_i .
- $\bar{\mathcal{A}}$ sends its state to the oracle which applies $\mathbb{I} \otimes \mathcal{O} \otimes \mathbb{I}_D$ to the state.
- $\bar{\mathcal{A}}$ applies the unitary $U_{q+1} \otimes \mathbb{I}_D$ and then measures the D registers.
- Finally, $\bar{\mathcal{A}}$ takes a measurement of the output registers.

Consider now the set of projections $\{\Pi_x^{D,i} : x \in X, i \in [q]\}$ where $\Pi_x^{D,i}$ denotes the projection of the i -th slot of the D registers onto the state $|x\rangle$. Also consider the set $\{\Pi_{\mathbf{x}}^D : \mathbf{x} \in X^q\}$ where $\Pi_{\mathbf{x}}^D$ is the projection of the D registers onto $|x_1, \dots, x_q\rangle$ from which it can be seen that

$$\Pi_{\mathbf{x}}^D = \Pi_{x_q}^{D,q} \dots \Pi_{x_1}^{D,1}.$$

Furthermore, recall that Π_x denotes the projection of the query input register onto $|x\rangle$. Then we have the following claim:

Claim. Take C_i to be the unitary from the description of $\bar{\mathcal{A}}$. For any value of $\mathbf{x} = (x_1, \dots, x_q) \in X^q$, any $|\phi_0\rangle$, any sequence of unitaries U_1, \dots, U_q and any oracle \mathcal{O} , the two following (un-normalized) states are equal:

1. $U_{q+1} \cdot C_q \cdot \Pi_{x_q} \cdot U_q \dots C_2 \cdot \Pi_{x_2} \cdot U_2 \cdot C_1 \cdot \Pi_{x_1} \cdot U_1 (|\phi_0\rangle \otimes |0, \dots, 0\rangle_D)$
2. $\Pi_{\mathbf{x}}^D \cdot U_{q+1} \cdot C_q \cdot U_q \dots C_2 \cdot U_2 \cdot C_1 \cdot U_1 (|\phi_0\rangle \otimes |0, \dots, 0\rangle_D)$

Proof. We first show that for any i , applying $C_i \cdot \Pi_{x_i}$ to a state where the i -th D register is 0 is the same as applying $\Pi_{x_i}^{D,i} \cdot C_i$ to that same state. Note that both C_i and Π_{x_i} are the identity on all registers other than the query input register and i -th D register. Therefore, ignoring all registers except for the query input and i -th D register, we have $C_i \cdot \Pi_{x_i} (|x\rangle \otimes |0\rangle_{D,i}) = \delta_{x,x_i} |x\rangle \otimes |x\rangle = \Pi_{x_i}^{D,i} \cdot C_i (|x\rangle \otimes |0\rangle_{D,i})$. Therefore the first state in the claim is equal to

$$U_{q+1} \Pi_{x_q}^{D,q} C_q U_q \dots \Pi_{x_2}^{D,2} C_2 U_2 \Pi_{x_1}^{D,1} C_1 U_1 (|\phi_0\rangle \otimes |0, \dots, 0\rangle_D).$$

Next, note that the projection $\Pi_{x_i}^{D,i}$ is the identity on all registers except for the i -th D register and that all matrices to the left of it are the identity on the i -th D register. Therefore, each $\Pi_{x_i}^{D,i}$ commutes with everything to the left of it, so the fact that

$$\Pi_{\mathbf{x}}^D = \Pi_{x_q}^{D,q} \dots \Pi_{x_1}^{D,1}$$

completes the proof. \square

Carefully examining the two states from the above claim, we can see that the first state corresponds to \mathcal{A} 's pre-output measurement state given intermediate measurement results \mathbf{x} (and copying classical information into the D registers along the way) and the second corresponds to $\bar{\mathcal{A}}$'s pre-output measurement given that the D registers are measured to contain \mathbf{x} . Therefore, for any given oracle-input measurement sequence \mathbf{x} , the resulting state of \mathcal{A} and $\bar{\mathcal{A}}$ are *identical*. What remains is to show that the distribution of oracle-input measurements of \mathcal{A} and $\bar{\mathcal{A}}$ is the same.

In the second state from the claim, all matrices applied to $(|\phi\rangle \otimes |0, \dots, 0\rangle)$ are unitary apart from the projection. Therefore, before the projection is applied, we have a *normalized* state, which implies that the probability that \mathcal{A} measure the D registers in the state \mathbf{x} is exactly

$$\left\| \Pi_{\mathbf{x}}^D \cdot U_{q+1} \cdot C_q \cdot U_q \dots C_2 \cdot U_2 \cdot C_1 \cdot U_1 (|\phi_0\rangle \otimes |0, \dots, 0\rangle_D) \right\|^2.$$

On the other hand, let us consider the probability when $q = 2$ i.e. \mathcal{A} just measures some x_1 and x_2 . The probability of measuring x_1 is $\left\| \Pi_{x_1} \cdot U_1 (|\phi_0\rangle \otimes |0, \dots, 0\rangle) \right\|^2$, and the probability of it measuring x_2 *given* that it measured x_1 is

$$\left\| \Pi_{x_2} \cdot U_2 \cdot C_1 \cdot \frac{\Pi_{x_1} \cdot U_1 (|\phi_0\rangle \otimes |0, \dots, 0\rangle)}{\left\| \Pi_{x_1} \cdot U_1 (|\phi_0\rangle \otimes |0, \dots, 0\rangle) \right\|} \right\|^2.$$

Therefore, the probability that \mathcal{A} measures (x_1, x_2) is

$$\left\| \Pi_{x_2} \cdot U_2 \cdot C_1 \cdot \Pi_{x_1} \cdot U_1 (|\phi_0\rangle \otimes |0, \dots, 0\rangle) \right\|^2.$$

Following this logic through for general q and noting that applying the unitary $U_{q+1} \cdot C_q$ does not affect norms, we have that the probability of \mathcal{A} measuring the sequence \mathbf{x} is

$$\left\| U_{q+1} \cdot C_q \cdot \Pi_{x_q} \cdot U_q \dots C_2 \cdot \Pi_{x_2} \cdot U_2 \cdot C_1 \cdot \Pi_{x_1} \cdot U_1 (|\phi_0\rangle \otimes |0, \dots, 0\rangle_D) \right\|^2,$$

which is precisely the same as $\bar{\mathcal{A}}$'s probability by the claim above.

E Separate Robustness and Reusability Proofs

Although plain robustness is implied by reusable robustness, we consider the former here to explicitly address the first key question from the introduction. This section also serves as intuition for the more complex robustly reusable proofs. We also stress that reusable robustness (proved in Sect. 4.2) also implies the stronger *post-application* version of robustness discussed in Sect. 2.

E.1 Robustness in the Classical ROM

Here we will write a relatively simple proof of robustness in the classical ROM. Note that the lack of explicit dependence of the security bound on the number of correctable errors (i.e. t) is achieved by considering *linear* secure sketches along with the robustness definition from Sect. 2. Once we have this proof, we can aim to translate it into the QROM setting using the O2H lemma and Zhandry's quantum query recording techniques as is done in the full version. Taking $k = \lambda$ and query bound $q = 2^\lambda$, the below shows that we may use a (m, m', t) secure sketch where $m' \approx 2\lambda$ to achieve λ bits of security in the classical ROM.

Theorem 4. *Let (SS, Rec) be a linear (m, m', t) secure sketch. Then the hash construction HFE is a robust $(\mathcal{M}, m, \ell, t, \epsilon = 2q \cdot 2^{-m'})$ fuzzy extractor with robustness $\delta \leq 2^{-k} + 2q \cdot 2^{-m'}$ in the classical ROM against unbounded adversaries making at most q queries to the random oracle.*

Proof. Assume that the adversary \mathcal{A} is unbounded and deterministic, but only accesses the oracle H at most q times. Throughout, we parse H as H_1 (the function that outputs the first k bits of H) and H_2 (the function that outputs the final ℓ bits of H). Denote the hint/helper data given to the adversary in the robustness game as $P := (s, h = H_1(w, s))$ where w is the secret value sampled by the challenger. It is assumed that w follows *any* any distribution \mathcal{W} with min-entropy at least m . We denote the algorithm $\mathcal{E}_{\mathcal{A}}^H(w, P)$ to be the robustness experiment that is played between the challenger and adversary \mathcal{A} with respect to the values w and P . Concretely, $\mathcal{E}_{\mathcal{A}}^H$ on input (w, P) is as follows:

1. Run $\mathcal{A}^H(P)$ and wait for it to output $(\Delta^*, P^* = (s^*, h^*))$.
2. Decide on the ultimate output according to the following:
 - If $\|\Delta^*\| > t$: output 0.
 - If $s^* = s$: output 0.
 - If $H_1(w + f(\Delta^*, s^*, s), s^*) \neq h^*$ (where f is from the linearity property of the sketch and H_1 is computed via a random oracle query): output 0. Otherwise output 1.

It can be seen that the probability that \mathcal{A} wins the robustness experiment is then $\delta = \Pr_{w,P,H}[1 \leftarrow \mathcal{E}_{\mathcal{A}}^H(w, P)]$ after noting that if $\|\Delta\| \leq t$ and $s^* = s$, it is required that $P^* = P$ for the hash value to be valid. Now suppose we sample a function G as follows: set $G(x) = H(x)$ for all $x \neq (w, s)$ and set $G(w, s)$ uniformly in $\{0, 1\}^k$. The point of introducing this function G is that it is independent of the input P given to the adversary. Consider the event \mathbb{E} that \mathcal{A} queries its oracle on the point (w, s) at some point during execution of $\mathcal{E}_{\mathcal{A}}$. Note that G and H are *identical* apart from on the input (w, s) . Therefore, the output of $\mathcal{E}_{\mathcal{A}}^G(P)$ and $\mathcal{E}_{\mathcal{A}}^H(P)$ are identical unless there is an oracle query on (w, s) . Note that $\mathcal{A}^G(P)$ queries its oracle on (w, s) if and only if $\mathcal{A}^H(P)$ does (assuming \mathcal{A} is deterministic). Also, any potential random oracle query performed when $\mathcal{E}_{\mathcal{A}}$ makes its output decision cannot possibly be (w, s) as it is guaranteed that $s^* \neq s$. Therefore, we can write

$$\left| \Pr_{w,P} [1 \leftarrow \mathcal{E}_{\mathcal{A}}^H(w, P)] - \Pr_{G,H} [1 \leftarrow \mathcal{E}_{\mathcal{A}}^G(w, P)] \right| \leq \Pr_{w,P} [\mathbb{E} : \mathcal{E}_{\mathcal{A}}^G(w, P)]. \quad (3)$$

$\mathcal{A}^G(P)$ is given input $P = (s, H_1(w, s))$ and access to G where $G(w, s)$ is independent of $H_1(w, s)$. This means that the uniform value $H_1(w, s)$ does not leak any information on w at all given access to G . Therefore, if $\mathcal{A}^G(P)$ makes q oracle queries, the probability that it queries (w, s) is at most $q \cdot 2^{-\gamma_s}$ where $\gamma_s = \tilde{H}_{\infty}(\mathcal{W} \mid s)$ for a fixed P . Taking an expectation over P , we find that $\Pr[\mathbb{E} : \mathcal{E}_{\mathcal{A}}^G(w, P)] \leq q \cdot 2^{-m'}$ by the security of the secure sketch.

Now we analyse $\Pr[1 \leftarrow \mathcal{E}_{\mathcal{A}}^G(w, P)]$. Denote the sequence of queries that \mathcal{A} submits to the oracle G as $Q = ((w_1, s_1), \dots, (w_q, s_q))$ (we parse each query as (w_i, s_i) and pad Q to make it have length exactly q). Recall that we use f to denote the function from the linearity condition on the secure sketch. If $(w + f(\Delta^*, s^*, s), s^*)$ was not in Q , then the value of $G(w + f(\Delta^*, s^*, s), s^*)$ is

uniform from \mathcal{A} 's perspective, in which case h^* is correct with probability 2^{-k} . Therefore, if this point is not in Q , then \mathcal{E}_A^G outputs 1 with probability 2^{-k} . On the other hand, suppose that $(w + f(\Delta^*, s^*, s), s^*)$ does appear in Q and consider the algorithm $\bar{\mathcal{A}}(s)$ that (i) takes input of the form $s = \text{SS.Gen}(w)$, (ii) samples $h \leftarrow \{0, 1\}^k$ and runs $\mathcal{A}^G((s, h))$ (simulating G) until it outputs (Δ^*, s^*, h^*) recording the query list Q , and (iii) for $i \leftarrow [|Q|]$, takes the i -th element of Q denoted by $x_i = (w_i, s_i)$ and outputs $w_i - f(\Delta^*, s^*, s)$. Now $\bar{\mathcal{A}}(s)$ outputs w when both $(w + f(\Delta^*, s^*, s), s^*)$ appears in Q and it chooses a correct value i . This implies that

$$\Pr_s[w \leftarrow \bar{\mathcal{A}}(s)] = \frac{1}{q} \cdot \Pr[(w + f(\Delta^*, s^*, s), s^*) \in Q].$$

Furthermore, the probability of any algorithm (including $\bar{\mathcal{A}}$) outputting w on input s is at most $2^{-\gamma s}$ for fixed s . Overall we have

$$\begin{aligned} \Pr_{\substack{w,P, \\ G,H}} [1 \leftarrow \mathcal{E}_A^G(w, P)] &\leq 2^{-k} + \Pr_{\substack{w,P, \\ G,H}} [(w + f(\Delta^*, s^*, s), s^*) \in Q] \\ &= 2^{-k} + q \cdot \Pr_s[w \leftarrow \bar{\mathcal{A}}(s)] \\ &\leq 2^{-k} + q \cdot 2^{-m'}. \end{aligned}$$

Combining the above inequality with Eq. (3) completes the proof for the robustness bound. For the bound on ϵ , we can say that the advantage of \mathcal{A} in distinguishing $R = H_2(w)$ from a uniform value U given $P = (s, h)$ is at most the probability that \mathcal{A} queries H on (w, s) given that it knows $P = (s, h)$. To upper bound this probability we can use Eq. (3), but instead of considering the event that \mathcal{E}_A^G or \mathcal{E}_A^H outputs 1, we consider the event that (w, s) is queried to the random oracle by \mathcal{A} during the execution of \mathcal{E}_A . The RHS of the inequality remains the same so we may reuse the analysis above to bound it. Also, similarly to the above, if $\mathcal{E}_A^G(w, P)$ outputs (w, s) in the query list to G , there must be an algorithm similar to $\bar{\mathcal{A}}$ (with input s) that finds w by picking some query made to G at random. \square

F Security Proof for Ring-LPN SKEM

Lemma 6. SKEM_{RLPN} is correct as long as $\tau m \leq t$.

Proof. Take any encapsulation key $\mathbf{k}_1 \in R_2^k, k_2 \in \mathbb{Z}_2^k$, and encapsulation randomness $a \in R_2, \tilde{k} \in \mathbb{Z}_2^k$. Also consider \mathbf{e} sampled according to $\text{Ber}_\tau^{\kappa \cdot n}$. Then the ciphertext/output key is

$$(c_1, \mathbf{c}_2) := (a, a\mathbf{k}_1 + \mathbf{e} + \mathcal{E}(\tilde{k})), \quad k' = \tilde{k} \oplus k_2.$$

Note that for $\mathbf{e} \leftarrow \text{Ber}_\tau^{\kappa \cdot n}$, the probability that \mathbf{e} has Hamming weight larger than τm is negligible as in [16]. Therefore, $\mathcal{D}(\mathbf{c}_2 - c_1\mathbf{k}) = \mathcal{D}(e + \mathcal{E}(\tilde{k})) = \tilde{k}$ with all but negligible probability. \square

Lemma 7. *The SKEM described above has key-shift pseudorandomness as long as $\text{RLPN}_{\kappa,R,\tau}$ is hard.*

Proof. We will provide a reduction from an algorithm \mathcal{B} attempting to solve the $\text{RLPN}_{m',R,\tau}$ problem to the algorithm \mathcal{A} that plays the key-shift pseudorandomness game. We suppose that \mathcal{B} has access to an oracle that returns arbitrarily many pairs (a, \mathbf{b}) where (a, \mathbf{b}) is either from the RLPN distribution or from the uniform distribution. Using \mathcal{A} as a sub-routine, \mathcal{B} carries out the following steps:

1. \mathcal{B} samples the public parameters $\text{pp} \leftarrow \text{SKEM}_{\text{RLPN}}.\text{Setup}(\lambda)$ and gives them to \mathcal{A} . \mathcal{B} also samples k_2 .
2. Whenever \mathcal{A} makes a \mathcal{O}^{ksp} -query on $\delta = (\delta_1, \delta_2) \in R_2^\kappa \times \{0, 1\}^k$, \mathcal{B} begins by asking its RLPN challenge oracle for a sample (a, \mathbf{b}) . It then samples $\tilde{k} \leftarrow R_2^\kappa$ and returns

$$(c_1, \mathbf{c}_2) = (a, \mathbf{b} + a \cdot \delta_1 + \mathcal{E}(\tilde{k})), \quad k' = \tilde{k} \oplus (k_2 \oplus \delta_2)$$

to \mathcal{A} .

3. When \mathcal{A} has finished making its queries, it outputs a bit b' . \mathcal{B} outputs b' .

First note that if \mathcal{B} has access to truly uniform samples (a, \mathbf{b}) , then \mathcal{B} 's answers to \mathcal{A} 's oracle queries are truly uniform and independent. In other words, \mathcal{B} perfectly simulates the key-shift pseudorandomness experiment when $b = 0$. On the other hand, suppose that \mathcal{B} has access to samples of the form $(a, a\mathbf{s} + e)$ where $\mathbf{s} \leftarrow R_2^\kappa$ that is fixed across queries. In this case, \mathcal{B} 's answers take the form

$$(c_1, \mathbf{c}_2) = (a, a(\mathbf{s} + \delta_1) + \mathbf{e} + \mathcal{E}(\tilde{k})), \quad k' = \tilde{k} \oplus (k_2 \oplus \delta_2)$$

which is precisely the required form when $b = 1$ in the key-shift pseudorandomness experiment. Therefore, if \mathcal{B} outputs the same as \mathcal{A} , then the advantage of \mathcal{B} in the $\text{RLPN}_{\kappa,R,\tau}$ problem is exactly the advantage of \mathcal{A} in the key-shift pseudorandomness game. \square

The decapsulation uniformity proof follows trivially from the discussion in Remark 2.

References

1. Alamélou, Q., et al.: Pseudoentropic isometries: a new framework for fuzzy extractor reusability. In: AsiaCCS (2018)
2. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. In: USENIX (2016)
3. Ambainis, A., Hamburg, M., Unruh, D.: Quantum security proofs using semiclassical oracles. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11693, pp. 269–295. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_10
4. Apon, D., Cho, C., Eldefrawy, K., Katz, J.: Efficient, reusable fuzzy extractors from LWE. In: Dolev, S., Lodha, S. (eds.) CSCML 2017. LNCS, vol. 10332, pp. 1–18. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60080-2_1

5. Becker, G.T.: Robust fuzzy extractors and helper data manipulation attacks revisited: theory vs practice. Cryptology ePrint Archive, Report 2017/493 (2017)
6. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_3
7. Boyen, X.: Reusable cryptographic fuzzy extractors. In: CCS (2004)
8. Boyen, X., Dodis, Y., Katz, J., Ostrovsky, R., Smith, A.: Secure remote authentication using biometric data. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 147–163. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_9
9. Canetti, R., Fuller, B., Paneth, O., Reyzin, L., Smith, A.: Reusable fuzzy extractors for low-entropy distributions. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 117–146. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_5
10. Cheon, J.H., Jeong, J., Kim, D., Lee, J.: A reusable fuzzy extractor with practical storage size: modifying Canetti *et al.*'s construction. In: Susilo, W., Yang, G. (eds.) ACISP 2018. LNCS, vol. 10946, pp. 28–44. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93638-3_3
11. Cramer, R., Dodis, Y., Fehr, S., Padró, C., Wichs, D.: Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 471–488. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_27
12. Cui, N., Liu, S., Gu, D., Weng, J.: Robustly reusable fuzzy extractors with imperfect randomness. Des. Codes Cryptogr. (2021)
13. Dodis, Y., Kanukurthi, B., Katz, J., Reyzin, L., Smith, A.: Robust fuzzy extractors and authenticated key agreement from close secrets. Cryptology ePrint Archive, Report 2010/456 (2010)
14. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: how to generate strong keys from biometrics and other noisy data. In: SIAM (2008)
15. Fuller, B., Meng, X., Reyzin, L.: Computational fuzzy extractors. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8269, pp. 174–193. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42033-7_10
16. Gilbert, H., Robshaw, M.J.B., Seurin, Y.: How to encrypt with the LPN problem. In: ICALP (2008)
17. Heyse, S., Kiltz, E., Lyubashevsky, V., Paar, C., Pietrzak, K.: Lapin: an efficient authentication protocol based on ring-LPN. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 346–365. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34047-5_20
18. Holcomb, D.E., Bursleson, W.P., Fu, K.: Power-up SRAM state as an identifying fingerprint and source of true random numbers. IEEE Trans. Comput. **58**(9) (2009)
19. Huth, C., Becker, D., Guajardo, J., Duplys, P., Güneysu, T.: Securing systems with scarce entropy: LWE-based lossless computational fuzzy extractor for the IoT. Cryptology ePrint Archive, Report 2016/982 (2016)
20. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1
21. Marinissen, E.J., et al.: IoT: source of test challenges. In: IEEE European Test Symposium (ETS) (2016)
22. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC (2005)

23. Unruh, D.: Revocable quantum timed-release encryption. *J. ACM* (2015)
24. Wen, Y., Liu, S.: Reusable fuzzy extractor from LWE. In: Susilo, W., Yang, G. (eds.) ACISP 2018. LNCS, vol. 10946, pp. 13–27. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93638-3_2
25. Wen, Y., Liu, S.: Reusable fuzzy extractor from the decisional Diffie-Hellman assumption. *Des. Codes Cryptogr.* (2018)
26. Wen, Y., Liu, S.: Robustly reusable fuzzy extractor from standard assumptions. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 459–489. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_17
27. Wen, Y., Liu, S., Gu, D.: Generic constructions of robustly reusable fuzzy extractor. In: Lin, D., Sako, K. (eds.) PKC 2019. LNCS, vol. 11443, pp. 349–378. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17259-6_12
28. Zhandry, M.: How to record quantum queries, and applications to quantum indistinguishability. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11693, pp. 239–268. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_9



Subversion-Resilient Authenticated Encryption Without Random Oracles

Pascal Bemmam¹(✉), Sebastian Berndt², Denis Diemert¹,
Thomas Eisenbarth², and Tibor Jäger¹

¹ Bergische Universität Wuppertal, Wuppertal, Germany
{bemmam, diemert, tibor.jager}@uni-wuppertal.de

² Universität zu Lübeck, Lübeck, Germany
{s.berndt, thomas.eisenbarth}@uni-luebeck.de

Abstract. In 2013, the Snowden revelations have shown subversion of cryptographic implementations to be a relevant threat. Since then, the academic community has been pushing the development of models and constructions to defend against adversaries able to arbitrarily subvert cryptographic implementations. To capture these strong capabilities of adversaries, Russell, Tang, Yung, and Zhou (CCS'17) proposed CPA-secure encryption in a model that utilizes a trusted party called a *watchdog* testing an implementation before use to detect potential subversion. This model was used to construct subversion-resilient implementations of primitives such as random oracles by Russell, Tang, Yung, and Zhou (CRYPTO'18) or signature schemes by Chow et al. (PKC'19) but primitives aiming for a CCA-like security remained elusive in any watchdog model. In this work, we present the first subversion-resilient authenticated encryption scheme with associated data (AEAD) without making use of random oracles. At the core of our construction are subversion-resilient PRFs, which we obtain from weak PRFs in combination with the classical Naor–Reingold transformation. We revisit classical constructions based on PRFs to obtain subversion-resilient MACs, where both tagging and verification are subject to subversion, as well as subversion-resilient symmetric encryption in the form of stream ciphers. Finally, we observe that leveraging the classical Encrypt-then-MAC approach yields subversion-resilient AEAD. Our results are based on the trusted amalgamation model by Russell, Tang, Yung, and Zhou (ASIACRYPT'16) and the assumption of honest key generation.

Keywords: Subversion · Authenticated Encryption · Symmetric Cryptography

1 Introduction

While many cryptographic primitives nowadays have sound security proofs based on widely believed complexity-theoretic assumptions *implementing* these primitives securely is highly non-trivial as many possible attacks are not captured by the formal security models. For example, a malicious party can intentionally

embed a *covert channel* into an implementation of a cryptographic application. This kind of subversion is widely known as *kleptography* or *algorithm substitution attacks* (ASAs) and was first studied by Young and Yung [42].

The most prominent real-world example is the modification of the `Dual_EC` pseudorandom number generator [40] by the NSA. This pseudorandom generator involves two constants P and Q . If these constants are chosen independently, `Dual_EC` is secure [16], but the *designer* of the implementation (or the standard) can easily construct two dependent constants P and Q allowing them to reconstruct the state. We refer to the work of Checkoway et al. [19] for a more in-depth discussion. While the possibility of a backdoor due to [40] has been known since 2007, it was not known whether such backdoors would be used by law enforcement agencies such as the NSA. This changed with the revelation of internal NSA documents by Edward Snowden in 2013. These documents explicitly talk about *Project Bullrun*, where resources are used to “Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.” and “Influence policies, standards, and specifications for commercial public key technologies.” [34]. These revelations reignited the interest in subversion attacks with the work by Bellare, Paterson, and Rogaway [10]. In general, the subverter \mathcal{A} of a primitive Π has two roles: First, they provide a subverted implementation denoted by $\tilde{\Pi}$. Second, they participate in the usual security experiment and aim to break the security guarantees provided by the unsubverted implementation of Π .

1.1 Subversion-Resilience Models

While many different subversion attacks were studied in different scenarios, the universal stateless attack by Bellare, Jaeger, and Kane [7] showed that it is *impossible* to prevent subversion attacks against symmetric encryption schemes without additional assumptions. Later, this impossibility result was extended by Berndt and Liškiewicz [12] to also hold for all randomized algorithms. To circumvent these impossibility results different models were used. Here, we focus on the most prominent models, which are *cryptographic reverse firewalls*, *self-guarding schemes*, the *immunization model*, and the *watchdog model*. We emphasize that these models seem incomparable as they rely on different assumptions. Each use case requires a careful investigation of which model reflects the considered setting best.

Cryptographic Reverse Firewalls. Mironov and Stephens-Davidowitz [32] introduced *cryptographic reverse firewalls*. In this model a third party called *firewall* resides between the communicating parties involved in a cryptographic protocol. These firewalls “sanitize” the communication between the parties, usually by rerandomizing the messages sent by the parties. As the firewalls are modeled as non-subverted algorithms, this prevents leakage of sensitive information by the subverted implementation. An important feature of these firewalls is the fact that they do *not* have access to the secret keys of the parties and do not “provide security”, i.e., they do not help non-subverted implementations in achieving security objectives. As described above, the typical approach for reverse firewalls

is to rerandomize the communication, see, for example, [4, 17, 18, 20, 25, 32]. Even though many cryptographic primitives allow for rerandomization, primitives that aim for authenticity do not allow this. To still guarantee subversion-resilience in the firewall model, one needs to revert to strong assumptions. For example, Mironov and Stephens-Davidowitz [32] either make use of a symmetric bilinear map where the *inverse computational Diffie–Hellman assumption*, which is strictly stronger than the computational Diffie–Hellman assumption [32], holds or need a *strongly rerandomizable* asymmetric encryption scheme, for which no candidate is currently known. This allows them to design IND-CCA-secure protocols. Alternatively, Bossuat et al. [15] equip the reverse firewall with a key shared with both endpoints it aims to protect, which deviates from the standard assumptions described above.

Self-Guarding Schemes. Fischlin and Mazaheri [27] introduced the notion of *self-guarding schemes* that split each scheme into an *initialization phase* and a *computation phase*. Here, the initialization phase is supposed to be unsubverted, and thus outputs produced during that phase can be used to sanitize the output of the possibly subverted computation phase.

Immunization. Dodis, Ganesh, Golovnev, Juels and Ristenpart [24] formalized an *immunization model* for pseudorandom generators, where an immunization function is applied to the output of the generator. Depending on the knowledge of the subverter about this function, the authors show different approaches for choose the immunization function in such a way that the output of the subverted pseudorandom generator is indistinguishable from the output of an honest pseudorandom generator. In both the *semi-private* and the *private* model, the authors were able to construct such functions.

Watchdog Model. Bellare, Paterson, and Rogaway [10] introduced a model in which a trusted monitoring party called *watchdog*¹ tests a primitive for subversion. Russell, Tang, Yung, and Zhou [37] introduced a non-black-box-variant of this model, called the *trusted amalgamation model*, where the designer of a primitive is allowed to split the primitive into different components that each can individually be checked the watchdog. The complete primitive is then amalgamated from these components by the *trusted* (i.e., not subverted) amalgamation function. This amalgamation function should thus be as simple as possible. For several cryptographic primitives subversion-resilient constructions were proposed, for example, trapdoor one-way permutations [37], pseudorandom generators [11, 37], symmetric IND-CPA-encryption schemes [38], hash functions [27], asymmetric IND-CPA-encryption schemes [11, 38], random oracles [5, 23, 39]², signature schemes [21, 37], and key encapsulation mechanisms [11].

This paper also uses the trusted amalgamation model, so let us take a closer look at important details of this model. First, the order of the quantification is important, as discussed by Russell, Tang, Yung, and Zhou [37]. In the following, fix some primitive Π . One possibility to define that Π is subversion-resilient is

¹ They introduced the concept of detecting subversion rather than a “watchdog”.

² Note that the proof in [39] contained an error that was later fixed in [13].

to demand that for each subverter \mathcal{A} , there is a watchdog WD such that either 1) the watchdog WD detects the subversion provided by \mathcal{A} or 2) the subverted implementation provided by \mathcal{A} does not weaken the security guarantees. While, at first glance, this model closely resembles the usual cryptographic security model, the watchdog now needs to depend on the adversary and, to protect against multiple different adversaries, *all* corresponding watchdogs need to be deployed. A much more desirable solution is to use a single *universal* watchdog WD such that every subverter \mathcal{A} will be detected by this *single* watchdog. In our work, we use simple universal watchdogs that only sample uniformly random inputs and check the possibly subverted implementation against the honest specification.

For simplicity, we assume throughout this work that the attacker always provides stateless implementations. Similarly to the approach by Russell, Tang, Yung, and Zhou, we can also allow *rewindable* stateful implementations [37, Rem. 2.5]. This means that the watchdog is allowed to rewind the state of the implementation and test it for various inputs starting from the same state. If these are not rewindable, time bombs as introduced by Fischlin and Mazaheri [27] are possible, which seem to be unpreventable by an universal offline watchdog.

1.2 Towards Subversion-Resilient Authenticated Encryption

There has been huge progress over the last years and for many cryptographic primitives it was shown how to construct them in a subversion-resilient manner. However, authenticated encryption (AE in the following) where both encryption and decryption are subject to subversion has not been achieved in an offline watchdog model. The main challenge in constructing subversion-resilient AE is protecting the decryption algorithm. This is due to input-trigger attacks, which cannot be avoided without additional assumptions (such as trusted operations or the trusted amalgamation model) or using heavy machinery such as random oracles in a model where the decryption algorithm is modeled as a black box algorithm.

Russell, Tang, Yung, and Zhou [39] showed how to make random oracles subversion-resilient. This then leads to subversion-resilient signatures by Chow et al. [21], where both the signing and verification algorithm can be subverted while heavily relying on the subversion-resilient random oracle. The authors also showed how to construct subversion-resilient signatures in the standard model, where key generation and signing are subverted. While making use of random oracles may also directly lead to subversion-resilient AE, this work explores the possibility of achieving this notion *without* resorting to random oracles.

Another approach to fix subversion without random oracles was proposed by Ateniese, Francati, Magri, and Venturi [3]. Here, the authors show how to sanitize deterministic algorithms where all algorithms can be subverted, including the sanitizer. They present a transformation from an arbitrary algorithm to a subversion-resilient one. Their approach uses a secret (but tamperable) random source to generate the keys and the public parameters. However, the results in [3] only apply to deterministic primitives, for which it is well-known that they do

not achieve CPA security. In a setting where the adversary is allowed to freely choose inputs to its oracles, referred to as unconstrained games in [3], subversion-resilience is achieved by using an online watchdog, i.e., a watchdog which has access to transcripts of the considered security experiment. We, on the other hand, focus on offline watchdogs, which only get oracle access to the subverted algorithms *before* the security experiment is executed. As the construction of a subversion-resilient MAC where both the tagging and the verification algorithms may be subverted and the adversary chooses the input to its oracle is a crucial building block in our work, the results of [3] cannot be applied. Finally, Armour and Poettering showed in a series of works [1, 2] several attacks on decryption of AEAD and verification of message authentication. Similar approaches are used by Russell, Tang, Yung, and Zhou [38] who effectively rerandomize the inputs to subverted algorithms. Unfortunately, this cannot be directly applied to decryption/verification, as input trigger attacks cannot be avoided without assuming that rerandomization is done as a trusted operation.

Hence, we deduced the following main research question of this work:

Is it possible to construct subversion-resilient authenticated encryption without random oracles in an offline watchdog model while only assuming non-cryptographic building blocks to be trusted?

In this paper, we answer this question affirmatively.

On the Difficulties of Constructing Subversion-Resilient AE. Before we describe our solution, it is instructive to understand and recognize the difficulties in constructing subversion-resilient authenticated encryption. The first major obstacle lies in the existence of *input trigger attacks*, first formalized by Degabriele, Farshim, and Poettering [22]: Such an attack modifies the underlying algorithm only on a single, arbitrary input x^* called the trigger. Whenever the algorithm is given x^* as input, it deviates from the specification by, e.g., outputting the secret key. As these triggers are chosen randomly by the attacker, no offline watchdog can detect the presence of these triggers. Thus, Degabriele, Farshim, and Poettering proposed a solution using an online watchdog. Now, these triggers are naturally connected to security experiments that model a *search problem*. Namely, in the final communication step of these experiments, the adversary usually sends some input which is directly evaluated by the underlying primitive. This direct transfer of information from the attacker to the primitive leads to trigger attacks, as the attacker can simply choose to submit such a trigger that solves the search problem. Now, security experiments that aim to secure the authenticity of information are typically modeled as search problems, where the task of the attacker is to produce some kind of forgery. Hence, such primitives are quite vulnerable to trigger attacks. Furthermore, even if one only considers *decision problems*, a direct transfer of information from the attacker to the primitive still allows for the use of input triggers.

1.3 Our Contribution

In this work, we show that subversion-resilient AE can be achieved without the use of random oracles. We use the trusted amalgamation model proposed by Russell, Tang, Yung, and Zhou [37,38], which also inspired our work. To overcome input triggers, we need to avoid search problems and primitives that take direct input from the adversary. At the core of our construction are weak PRFs, i.e., PRFs that are only indistinguishable from random only if evaluated on *random* inputs. Our main contributions can be summarized as follows.

Weak PRFs are Subversion-Resilient. We first observe that these weak PRFs are naturally subversion-resilient, i.e., *any* implementation is indistinguishable from random given it passes the watchdog’s check.

Subversion-Resilient PRFs from Weak PRFs. As a next step, we use the classical Naor–Reingold transformation [33] that transforms a weak PRF into a PRF that can be queried *arbitrarily*. We prove that the Naor–Reingold construction also transforms a subversion-resilient weak PRF into a subversion-resilient PRF. In the context of subversion and the trusted amalgamation model, the trusted amalgamation applying the Naor–Reingold transformation can thus be seen as a *trusted data structure*, as the adversarially chosen inputs are only used to choose random keys in a trusted manner.

Subversion-Resilient AE from PRFs. Given subversion-resilient PRFs, the classical “PRF-as-MAC” approach also guarantees subversion-resilience by assuming canonical verification and a trusted comparison operation. Making use of subversion-resilient PRFs again, we show that the classical randomized counter mode is also subversion-resilient assuming a trusted \oplus operation, which can even be generalized to stream ciphers. From both a subversion-resilient MAC and encryption scheme, we then prove that subversion-resilient authenticated encryption can be achieved via the classical “Encrypt-then-MAC” approach.

1.4 Discussion

Finally, we discuss our contribution and the assumptions we make in this work.

Subversion-Resilient AEAD from ROs. Note that given a subversion-resilient RO, as proposed in [39], one could replace the subversion-resilient PRF in our work by the RO and would obtain the similar results. However, argueably a subversion-resilient RO is a much stronger assumption than the existence of a weak PRF (in the standard model) that we base our results on in this work. To obtain a subversion-resilient RO, Russell et al. also make use of a trusted XOR operation and thus also need the same trust-assumptions as we do in our work.

Previous Work on Subversion-Resilient MACs. In previous work, Fischlin, Janson, and Mazaheri [26] also observed that weak PRFs are a helpful tool to defend against backdoors. They show that a backdoored weak PRF implies a public key encryption scheme, arguing that the difference in performance can be easily detected. Further, they show that applying the randomized cascade (RC) construction by Maurer and Tessaro [31] to a weak PRF immunizes HMAC

against backdoors. As discussed later, using the RC construction also works for our construction, but requires to model the used prefix-free encoding as a trusted building block. Also, while Fischlin, Janson and Mazaheri focus on the properties of HMAC as a PRF, we focus on the subversion-resilience property of a MAC and its role in the Encrypt-then-MAC approach. As our model does not include detection based on the performance time of the subverted algorithm, we base the security of our construction on the subversion-resilience of weak PRFs.

Honest Key Generation. Contrary to previous works [21, 38] we dismiss modeling subversion of key generation as this is typically only an *abstraction* for some means to derive a secure key. Even if one can construct key generation that could be executed by a single party, it is not clear how both parties would end up with the same key, potentially requiring a secure channel for key transportation. But to do this, both parties need to participate in a key-exchange protocol, which usually allows for a wide range of possible subversions (see, e.g., [25, 32]). Hence, our work can be extended by some approach to derive uniform keys.

Weak PRFs. One may think that using weak PRFs instead of “standard” PRFs may be sufficient. However, it is not clear how to obtain MACs from (subversion-resilient) weak PRFs, as the security of MACs against forgery attacks are modeled as a search problem. While we would be able to answer all tagging queries via random queries to the PRF (e.g., via a Carter-Wegman-style [41] construction), handling the final forgery query is a challenge, as this query is directly made on the verification algorithm.

Trusted Operations. We make use of several trusted operations, as it is not hard to see that some sort of trusted operations are needed to avoid trigger attacks proposed in previous works [1, 2, 22]. While being necessary, we aimed to minimize the number of trusted operations. Our approach only uses a trusted comparison and a trusted XOR. We believe that both of these (non-cryptographic operations) are simple enough to be either regarded as trusted or realized in hardware in a trusted manner. Not using a trusted XOR operation would most likely imply the need for some sort of rerandomization of ciphertexts before decrypting to remove biases. To the best of our knowledge, there is no AE scheme fulfilling such a property. Further, a trusted comparison seems unavoidable as otherwise a verification or decryption algorithm could reject or accept chosen inputs (since an adversarially chosen input is fed into a subverted component), as input triggers are again possible.

Relation to Immunized PRGs. As described above, Dodis et al. [24] constructed subversion-resilient pseudorandom generators in both the *semi-private* and the *private* immunization model. We believe that classical constructions of PRFs from PRGs such as the one due to Goldreich, Goldwasser, and Micali [28] can be used to also obtain PRFs in the immunization model. But, while our constructions rely on an offline watchdog and the amalgamation assumption, the constructions in the immunization model rely on the fact that parts of the implementation (i.e., the immunization function) is hidden from the subverter. These assumptions are orthogonal to each other.

2 Subversion-Resilience

In this section, we define the notion of subversion-resilience and loosely follow the approach by Russell, Tang, Yung, and Zhou [37]. Before we define the actual notions, we first describe our general notation and the overall setting.

2.1 Notation and Model

Notation. Recall that we need to distinguish between the *specification* of a primitive Π and the *implementation* of Π provided by the adversary. To make the distinction between an honest specification of a primitive Π and a (possibly) subverted implementation more explicit, we use the following notation throughout this paper. We denote by $\widehat{\Pi}$ the specification of the primitive and by $\widetilde{\Pi}$ the implementation of that primitive provided by the adversary.

A *security experiment* Exp for a cryptographic primitive Π with security objective GOAL involves one party, namely the adversary \mathcal{A} trying to break the security objective against $\widehat{\Pi}$. In contrast, *subversion experiment* ExpSR is executed with an implementation of the considered primitive by the adversary and consists of three phases involving two parties: In the first phase, the *adversary* \mathcal{A} provides provides a subverted implementation $\widetilde{\Pi}$. This implementation is then examined by a *watchdog* WD that tries to detect the subversion in the second phase. Finally, in the third phase, the adversary \mathcal{A} takes part in the security experiment, where the subverted implementation is used. In the following, we always treat \mathcal{A} as pair $(\mathcal{A}_0, \mathcal{A}_1)$, where \mathcal{A}_0 provides the subverted implementation $\widetilde{\Pi}$ and \mathcal{A}_1 takes part in the security experiment. As usual, we denote the security parameter by λ .

Amalgamation. As discussed earlier, preventing subversion attack in a purely black-box way is not possible, as universal undetectable attacks are known, e.g., by Berndt and Liškiewicz [12]. Russell, Tang, Yung, and Zhou [37] thus introduced a non-black-box model called the *trusted amalgamation model*. While a primitive $\Pi = (\Pi_1, \dots, \Pi_r)$ usually consists of a few different algorithms, the trusted amalgamation model splits all of these components into *subroutines*. For example, an encryption scheme usually consists of $r = 3$ algorithms (KGen, Enc, Dec), but these might be composed of several subroutines used in different places. The trusted amalgamation model makes the use of these subroutines more explicit by representing a primitive as the list of subroutines $\pi = (\pi_1, \dots, \pi_n)$ and a *trusted amalgamation function* Am that takes this list and produces the algorithms corresponding to the primitive. Let us get back to the example given above, $\text{Am}(\pi) = (\text{KGen}, \text{Enc}, \text{Dec})$ that consists of subroutines π_i . We will allow the subverter to individually subvert the subroutines π_i arbitrarily by providing implementations $\widetilde{\pi}_i$, but assume that the amalgamation function is not subject to subversion. The security experiment is then played on $\widetilde{\Pi} = \text{Am}(\widetilde{\pi})$. This assumption is usually justified by making this amalgamation function as simple as possible such that it can be checked automatically. For example, the amalgamation in the construction of Bemann, Chen, and Jager [11] and in the

constructions of Russell, Tang, Yung, and Zhou [38] only handles inputs and outputs of different subroutines and makes use of a few XOR operations. In the following, we thus represent the specification \widehat{H} of a primitive as $\widehat{H} = (\text{Am}, \pi)$, where $\pi = (\pi_1, \dots, \pi_n)$ is the list of subroutines. We also need to consider the amalgamation function for a *single* algorithm Π_i of a primitive, which we denote by Am_i . That is, the amalgamation $\text{Am}(\pi) = (\text{Am}_1(\pi), \dots, \text{Am}_r(\pi))$ actually consists of a vector of amalgamation functions such that there is a function for each algorithm of the primitive. As a shortcut, we simply write $(\text{Am}, \widehat{\Psi})$ if a construction uses a subversion-resilient $\widehat{\Psi} = (\text{Am}_\Psi, \psi)$ as a building block.

Split-Program Model. In addition to trusted amalgamation, Russell, Tang, Yung, and Zhou [37] also used the *split-program* methodology. Similar to modern programming techniques, it is assumed that randomness generation is split from a randomized algorithm. The randomness generator and the deterministic algorithm can then be tested individually by the watchdog. We also use this methodology in our work.

Randomness Generation. The constructions in this paper rely on “good” (i.e., in particular trusted) randomness being available. For this, either one of the constructions proposed by Russell, Tang, Yung, and Zhou [38] or Bemmam, Chen, and Jager [11] can be used. Both works contain constructions generating randomness that is indistinguishable from random for the adversary providing the implementation *without* using random oracles. For this paper, we can use both constructions. Hence, for simplicity, we abstract away the randomness generation and assume that our constructions generate uniformly random bits, while being able to test randomized algorithm on selected random coins. This assumption allows us to simplify notation and focus on our contributions to enable authentication in the presence of subversion.

2.2 Subversion-Resilience

Next, we define the notion of subversion-resilience. Intuitively, we extend a “conventional” security experiment Exp by a preceding check for subversion of the primitive. Afterwards, the security experiment is executed. This is illustrated in Fig. 1. As we study both decision (i.e. indistinguishability) and search (i.e. unpredictability) problems in this paper, we associate with experiment Exp a “baseline win probability” denoted by δ that gives the winning probability of a naive attacker, i.e., $\delta = 0$ for search problems and $\delta = 1/2$ for decision problems. To extend Exp , we first run \mathcal{A}_0 to obtain a subverted implementation $\tilde{\pi}$ (Fig. 1, l. 1). The watchdog WD then tests the implementation before we run the security experiment Exp with adversary \mathcal{A}_1 on the subverted implementation $\tilde{H} = \text{Am}(\tilde{\pi})$ as usual (Fig. 1, l. 3). The variable `state` is only used to synchronize \mathcal{A}_0 and \mathcal{A}_1 . Throughout this work we use the convention that the watchdog outputs “true” in the case that subversion is detected. To formalize subversion-resilience, consider the next definition and the corresponding security experiment shown in Fig. 1.

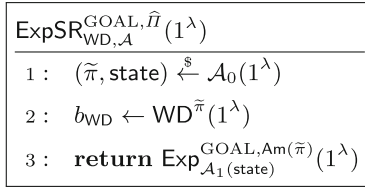


Fig. 1. The security experiment for GOAL-security under subversion.

Definition 1. A specification of a primitive $\widehat{\Pi} = (\text{Am}, \pi)$ is GOAL-secure under subversion in the offline watchdog model with trusted amalgamation if one can efficiently construct a ppt watchdog algorithm WD such that for any ppt adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ it holds

$$\text{AdvSR}_{\mathcal{A}}^{\text{GOAL}, \widehat{\Pi}}(1^\lambda, \delta) \text{ is negligible or } \text{Det}_{\text{WD}, \mathcal{A}}(1^\lambda) \text{ is non-negligible}$$

where $\text{AdvSR}_{\mathcal{A}}^{\text{GOAL}, \widehat{\Pi}}(1^\lambda, \delta) = |\Pr[\text{ExpSR}_{\text{WD}, \mathcal{A}}^{\text{GOAL}, \widehat{\Pi}}(1^\lambda) = 1] - \delta|$ and $\text{Det}_{\text{WD}, \mathcal{A}}(1^\lambda) = |\Pr[\text{WD}^{\tilde{\pi}}(1^\lambda) = 1] - \Pr[\text{WD}^\pi(1^\lambda) = 1]|$ using the experiment shown in Fig. 1, with $\delta \in \{0, \frac{1}{2}\}$ indicating whether a search or a decision problem is considered.

Note that $\text{Adv}_{\mathcal{A}}^{\text{GOAL}, \widehat{\Pi}}(1^\lambda, \delta)$ is not parameterized by the watchdog WD . We chose this approach to simplify notation, as the testing of the watchdog does not influence the advantage of the adversary directly. For public key encryption, this model is not equivalent to the model proposed by Russell, Tang, Yung, and Zhou [38], where the adversary has access to a subverted encryption oracle. For symmetric encryption, our more general definition captures theirs with some differences in syntax.

As mentioned earlier, we assume stateless subversion and that key and randomness generation are trusted. In order to shorten notation, we call primitives just GOAL-secure under subversion.

2.3 Achieving Subversion-Resilience

To prove our upcoming PRF construction subversion-resilient, we use an observation made by Russell, Tang, Yung, and Zhou [37]. If a deterministic primitive is only given inputs according to a *public* distribution and the implementation deviates from the specification with some probability δ (with inputs chosen according to this public input distribution), then a ppt watchdog can detect this with probability at least δ . Hence, in order to stay undetected, the number of inputs the implementation deviates from the specification needs to be negligible.

Lemma 1. Consider an implementation $\widetilde{\Pi} := (\widetilde{\pi}_1, \dots, \widetilde{\pi}_k)$ of a specification $\widehat{\Pi} = (\widehat{\pi}_1, \dots, \widehat{\pi}_k)$, where π_1, \dots, π_k are deterministic algorithms. Additionally, for each security parameter λ , public input distributions $X_\lambda^1, \dots, X_\lambda^k$ are defined respectively. If there exists a $j \in [k]$ such that $\Pr[\widetilde{\pi}_j(x) \neq \widehat{\pi}_j(x) : x \xleftarrow{\$} X_\lambda^j] = \delta$, this can be detected by a ppt offline watchdog with probability at least δ .

$\text{Exp}_{\mathcal{A},F}^{\text{PR}}(1^\lambda)$	$\text{Exp}_{\mathcal{A},F}^{\text{wPR}}(1^\lambda)$
1: $b \xleftarrow{\$} \{0,1\}; K \xleftarrow{\$} \mathcal{K}_\lambda$	1: $b \xleftarrow{\$} \{0,1\}; K \xleftarrow{\$} \mathcal{K}_\lambda$
2: if $b = 1$ then $b' \xleftarrow{\$} \mathcal{A}^{F(K,\cdot)}(1^\lambda)$	2: if $b = 1$ then $b' \xleftarrow{\$} \mathcal{A}^{(\$,F(K,\$))}(1^\lambda)$
3: else $g \xleftarrow{\$} \text{Func}(\mathcal{D}_\lambda, \mathcal{R}_\lambda); b' \xleftarrow{\$} \mathcal{A}^{g(\cdot)}(1^\lambda)$	3: else $g \xleftarrow{\$} \text{Func}(\mathcal{D}_\lambda, \mathcal{R}_\lambda); b' \xleftarrow{\$} \mathcal{A}^{(\$,g(\$))}(1^\lambda)$
4: return $b' == b$	4: return $b' == b$

Fig. 2. The security experiment for (weak) PRFs. Here, $\$$ denotes an input argument chosen uniformly at random from \mathcal{D}_λ upon any query issued by the adversary. Further, if $K \in \mathcal{K}_\lambda$, the oracle $F(K, \cdot)$ can only be queried on elements of \mathcal{D}_λ .

An instructive example to understand the usefulness of this lemma is the following. Suppose that we are given a single function f and a probability distribution X on the domain of f . In an experiment, the adversary can now issue a query, where $x \xleftarrow{\$} X$ is drawn and the pairs $(x, f(x))$ is given to the adversary. The goal of the adversary is to obtain a sample $(x^*, \tilde{f}(x^*))$, where $x^* \in X^*$ for some subset $X^* \subseteq \text{Supp}(X)$ such that $\tilde{f}(x^*) \neq \hat{f}(x^*)$ where $\text{Supp}(X)$ is the subset of values the variable X can take. Clearly, if the adversary can only perform a bounded number of samples, the density of X^* wrt. X cannot be arbitrarily small. But, as the distribution X is publicly known, a watchdog can also sample according to X and check the implementation \tilde{f} against the specification \hat{f} on these samples. Then, it is not hard to see that the adversary wins if the watchdog distinguishes the implementation from the specification.

3 Pseudorandom Functions

Intuitively, a PRF is a keyed function $F: \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$ associated with a key space \mathcal{K} , that is indistinguishable from a function sampled uniformly at random from the set of all functions $\mathcal{D} \rightarrow \mathcal{R}$. More formally, $\mathcal{K} = \bigcup_{\lambda \in \mathbb{N}} \mathcal{K}_\lambda$, $\mathcal{D} = \bigcup_{\lambda \in \mathbb{N}} \mathcal{D}_\lambda$, and $\mathcal{R} = \bigcup_{\lambda \in \mathbb{N}} \mathcal{R}_\lambda$. Additionally, we use $\text{Func}(\mathcal{D}, \mathcal{R})$ to denote the set of all functions mapping elements from \mathcal{D} to \mathcal{R} . In this paper, we only consider spaces that are subsets of $\{0,1\}^*$.³ Let us recall the standard definition of (weak) PRFs.

Definition 2. Let $T \in \{\text{wPR}, \text{PR}\}$ and let $\text{Exp}_{\mathcal{A},F}^T$ be defined as shown in Fig. 2. We define $\text{Adv}_{\mathcal{A},F}^T(1^\lambda) := |\Pr[\text{Exp}_{\mathcal{A},F}^T(1^\lambda) = 1] - 1/2|$. We say that F is pseudorandom if $\text{Adv}_{\mathcal{A},F}^{\text{PR}}(1^\lambda)$ is negligible for all ppt adversaries \mathcal{A} . Further, we say that F is weakly pseudorandom if $\text{Adv}_{\mathcal{A},F}^{\text{wPR}}(1^\lambda)$ is negligible for all ppt adversaries \mathcal{A} .

Weak PRFs are Subversion-Resilient. The first observation is that since all inputs given to the PRF are distributed uniformly at random, they follow a distribution that is publicly known. This allows us to apply Lemma 1. For an

³ We actually only require that we can sample uniform elements of \mathcal{D} and \mathcal{K} efficiently and that \mathcal{D} is a quasi group with operation \oplus .

implementation \tilde{F} of a specification \hat{F} of a weak PRF, let $\text{Neq}_\lambda \subseteq \mathcal{K}_\lambda \times \mathcal{D}_\lambda$ be the set of inputs, where \tilde{F} deviates from the specification, i.e., $\text{Neq}_\lambda = \{(K, x) \in \mathcal{K}_\lambda \times \mathcal{D}_\lambda \mid \tilde{F}(K, x) \neq \hat{F}(K, x)\}$. Now, consider the proportional amount p of Neq_λ , i.e., $p = \frac{|\text{Neq}_\lambda|}{|\mathcal{K}_\lambda \times \mathcal{D}_\lambda|}$. As the input distribution of the weak PRF experiment is public, Lemma 1 now directly implies the existence of a ppt watchdog with detection probability p (simply testing \tilde{F} on uniformly random inputs). Hence, for an adversary to succeed in the subversion-experiment, p must be negligible. But, as all inputs to the weak PRF are drawn randomly, the probability that an adversary making q queries will ever encounter an input to the weak PRF that belongs to Neq_λ is bounded by $q \cdot p$ and is thus negligible for ppt adversaries, as q is bounded by a polynomial in λ , yielding the following theorem.

Theorem 1. *If F is weakly pseudorandom, then the trivial specification $\hat{F} = F$ is weakly pseudorandom under subversion.*

3.1 Constructing Subversion-Resilient PRFs

In the following, we use the classical Naor–Reingold construction [33] to construct a (standard) PRF from a weak PRF that is subversion-resilient.

The Naor–Reingold Construction. Let $F_w: \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$ be a weak PRF. For the sake of simplicity, we only focus on the case that elements of \mathcal{K} , \mathcal{D} , and \mathcal{R} are of equal length and refer the reader to the survey by Bogdanov and Rosen [14] for generalizations. We now construct a (standard) PRF $F^{(\ell)}: \mathcal{K}^{2^\ell} \times \{0, 1\}^\ell \rightarrow \mathcal{R}$ that is parameterized by some integer ℓ of the form $\ell = 2^r$ describing the message length. It is easiest to construct $F^{(\ell)}$ inductively. In the simplest case of $\ell = 1$, the key of $F^{(\ell)}$ consists of two randomly sampled keys of F (i.e., two random bit strings $K_0, K_1 \in \mathcal{K}$). On input $x \in \{0, 1\}$, it returns K_x , i.e., $F^{(1)}((K_0, K_1), x) = K_x$. Given $F^{(\ell)}$, we construct $F^{(2\ell)}$ inductively as follows. A key of $F^{(2\ell)}$ consists of two keys $K_0^{(\ell)}$ and $K_1^{(\ell)}$ of $F^{(\ell)}$ (which in turn consists each of 2ℓ keys of F_w). On input $x = (x_1, x_2, \dots, x_{2\ell})$, the function $F^{(2\ell)}$ applies $F^{(\ell)}$ with the first key $K_0^{(\ell)}$ to the first half of x to obtain a key for F_w and then computes $F^{(\ell)}$ with the second key on the second half of x to obtain a value. More formally,

$$\begin{aligned} & F^{(2\ell)}((K_0^{(\ell)}, K_1^{(\ell)}), (x_1, \dots, x_{2\ell})) \\ &= F_w(F^{(\ell)}(K_0^{(\ell)}, (x_1, \dots, x_\ell)), F^{(\ell)}(K_1^{(\ell)}, (x_{\ell+1}, \dots, x_{2\ell}))). \end{aligned}$$

An useful alternate interpretation is the following (shown in Fig. 3). The key of $F^{(2\ell)}$ consists of 2ℓ key pairs $(K_{i,0}, K_{i,1})$ for $i = 1, \dots, 2\ell$. On input $(x_1, \dots, x_{2\ell})$, we construct a complete binary tree of height r , where $r = \log(2\ell)$. The final level of this binary tree contains the 2ℓ leaves. To produce the output of $F^{(2\ell)}$, we now construct a *labeling* of the vertices. We first label the i -th leaf of the tree with K_{i,x_i} , i.e., the message bit x_i determines whether we take $K_{i,0}$ or $K_{i,1}$. To obtain the label of an inner node v of the tree, we compute $F_w(\text{left}(v), \text{right}(v))$, where

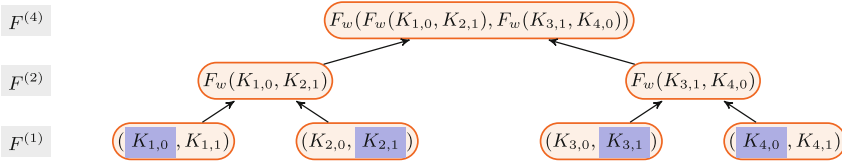


Fig. 3. The alternate interpretation of the Naor-Reingold construction as labeling of a complete binary tree for the value $x = (0, 1, 1, 0)$. The corresponding leaf values are marked in blue. (Color figure online)

left(v) (resp. right(v)) is the label of the left (resp. right) child of v . Finally, the output of $F^{(2^\ell)}$ is the label of the root of the tree. It is well-known that this construction gives a PRF $F^{(\ell)}$ if F_w is weakly pseudorandom.

Theorem 2 ([33, Thm. 5.1]⁴). *Let $\ell \in \mathbb{N}$ with $\ell = 2^r$. If F_w is weakly pseudorandom, then $F^{(\ell)}$ is pseudorandom.*

Now, observe that a non-subverted, honest call-structure to the underlying function F_w (which is trivially true due to our amalgamation assumption) directly implies the subversion-resilience of $F^{(\ell)}$. On the lowest level, $F^{(1)}$ will only return completely random values, which is clearly subversion-resilient. The inputs to $F^{(2)}$ are thus completely random values which follow a public input distribution and Lemma 1 directly implies subversion-resilience.

Theorem 3. *If \widehat{F} is pseudorandom under subversion, then for each ℓ with $\ell = 2^r$, $\widehat{F^{(\ell)}}$ is pseudorandom under subversion.*

Proof. Our watchdog simply samples random keys and random inputs for the weak PRF F_w and checks for deviations from the specification. As for the subversion-resilience of F_w discussed above, let Neq_λ be the set of inputs for which $\widehat{F_w}$ deviates from its specification. As shown before, by applying a watchdog that simply tests a sufficient number of random inputs to F_w , we know that the probability $p = \frac{|\text{Neq}_\lambda|}{|\mathcal{K}_\lambda \times \mathcal{D}_\lambda|}$ is negligible. On the lowest level, corresponding to $F^{(1)}$, we only choose one of two random values. Hence, the watchdog can easily verify the correctness of $F^{(1)}$ as there are only constantly many different inputs. In the next level, corresponding to $F^{(2)}$, the function F_w is only applied to these completely random inputs. If an adversary makes $q \in \text{poly}(\lambda)$ many queries, the probability that one of the calls to F_w on this level deviates from the specification is at most $q \cdot (\ell/2) \cdot p$, which is negligible. Conditioned on the event that all calls to F_w on the level corresponding to $F^{(2)}$ follow the specification, the inputs to the $q \cdot (\ell/4)$ calls to F_w on the level corresponding to $F^{(4)}$ are indistinguishable from random (due to the security of the specification of the weak PRF). Hence, with probability $q \cdot (\ell/4) \cdot p$, these inputs also do not belong to

⁴ Naor and Reingold use the notion of a *synthesizer*, which are in our context equivalent to weakly PRFS [14].

Neq_λ , if the inputs on the level corresponding to $F^{(2)}$ do not belong to Neq . Let $E_{\ell'}$ be the event that all inputs on the level corresponding to $F^{(\ell')}$ do not belong to Neq_λ . By iterating the above argumentation, it is not hard to see that $\Pr[E_{\ell'} \mid E_{\ell'/2}] \geq 1 - q \cdot (\ell/\ell') \cdot p$ holds. From $\Pr[E_2] \geq 1 - q \cdot (\ell/2) \cdot p$, we can conclude via a simple induction that

$$\begin{aligned} \Pr[E_{\ell'}] &= \Pr[E_{\ell'} \mid E_{\ell'/2}] \cdot \Pr[E_{\ell'/2}] + \Pr[E_{\ell'} \mid \neg E_{\ell'/2}] \cdot \Pr[\neg E_{\ell'/2}] \\ &\geq \prod_{i=1}^{r'} (1 - q \cdot (\ell/2^i) \cdot p) \end{aligned}$$

for $\ell' = 2^{r'}$. Hence, all probabilities $\Pr[E_{\ell'}]$ are of the form $1 - \text{negl}(\lambda)$ for a negligible function negl . We can thus conclude that the probability that any input to F_w belongs to Neq_λ is negligible. The original security guarantee due to Theorem 2 then directly implies the subversion-resilience of $F^{(\ell)}$. \square

Alternative Constructions. In principle any transformation from weak to standard PRFs can be used in our construction. We chose the Naor-Reingold construction, due to its simplicity and as it only requires the amalgamation function to act as a trusted data structure and no trusted operations. Alternatively, the randomized cascade construction by Maurer and Tessaro [31] can be used. There adversarially chosen messages are directly fed into a prefix-free encoding, which then needs to be modeled as a trusted operation in order to prevent input triggers. Another alternative is the IC construction by Maurer and Sjödin [30], where the input provided by the adversary is processed bitwise and either a weak PRF is executed or a previously computed value is used in an iterative process.

4 MAC

We now show how to construct subversion-resilient MACs using any subversion-resilient PRFs, e.g., the one from the previous section. Let us first recall the standard definition of MACs. A MAC works on a keyspace \mathcal{K} , message space \mathcal{M} , and tag space \mathcal{T} . Also, we have $\mathcal{K} = \bigcup_{\lambda \in \mathbb{N}} \mathcal{K}_\lambda$, $\mathcal{M} = \mathcal{M}_{\lambda \in \mathbb{N}}$, and $\mathcal{T} = \bigcup_{\lambda \in \mathbb{N}} \mathcal{T}_\lambda$.

Definition 3. We call a triple $\text{MAC} = (\text{KGen}, \text{Tag}, \text{Vf})$ a message authentication code (MAC) for key space \mathcal{K} , message space \mathcal{M} , and tag space \mathcal{T} . The randomized key generation algorithm KGen produces upon the security parameter 1^λ as input a key $K \xleftarrow{\$} \mathcal{K}_\lambda$. The randomized tagging algorithm Tag is given a key $K \in \mathcal{K}_\lambda$ and a message $M \in \mathcal{M}_\lambda$ and returns a tag $T \in \mathcal{T}_\lambda$. The deterministic verification algorithm Vf is given a key K , a message M , and a tag T and returns a bit b .

For correctness, we require that for all $K \in \mathcal{K}_\lambda$, for all $M \in \mathcal{M}_\lambda$, and all $T \in \text{Supp}(\text{Tag}(K, M))$, it holds $\text{Vf}(K, T) = 1$. Next, we recall the standard security notion of (strong) unforgeability of MACs.

Definition 4. Let MAC be a MAC and let $\text{Exp}_{\mathcal{A}, \text{MAC}}^{\text{SUF-CMA}}(1^\lambda)$ be defined as shown in Fig. 4. We define $\text{Adv}_{\mathcal{A}, \text{MAC}}^{\text{SUF-CMA}}(1^\lambda) := \Pr[\text{Exp}_{\mathcal{A}, \text{MAC}}^{\text{SUF-CMA}}(1^\lambda) = 1]$ and say that MAC is strongly unforgeable under a chosen message attack, or *SUF-CMA-secure*, if $\text{Adv}_{\mathcal{A}, \text{MAC}}^{\text{SUF-CMA}}(1^\lambda)$ is negligible for all ppt adversaries \mathcal{A} .

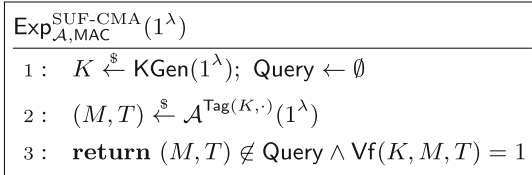


Fig. 4. The forgery experiment for MACs. On input $M \in \mathcal{M}_\lambda$ the oracle $\text{Tag}(K, \cdot)$ computes $T \xleftarrow{\$} \text{Tag}(K, M)$, stores (M, T) in Query and returns T .

4.1 MAC from PRFs

Consider the following generic construction of a (fixed-length) deterministic MAC based on a PRF. Let F be a keyed function $F : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$. We define $\text{MAC}_F = (\text{KGen}_F, \text{Tag}_F, \text{Vf}_F)$ with key space \mathcal{K} , message space \mathcal{M} , and tag space \mathcal{T} such that KGen_F on input 1^λ outputs a uniform key $K \xleftarrow{\$} \mathcal{K}_\lambda$, Tag_F on input key $K \in \mathcal{K}_\lambda$ and message $M \in \mathcal{D}_\lambda$, returns $T = F(K, M)$, and Vf_F on input a key $K \in \mathcal{K}_\lambda$, message $M \in \mathcal{D}_\lambda$, and a tag $T \in \mathcal{R}_\lambda$, outputs 1 if and only if $T = \text{Tag}_F(K, M)$. It is a well-known result by Goldreich, Goldwasser and Micali that a PRF can directly be used to construct a MAC which we recall.

Theorem 4 ([29]). *If F is pseudorandom, then MAC_F is SUF-CMA-secure.*

Theorem 4 guarantees that the subversion-resilience of the underlying function F directly transfers to MAC_F , if the $=$ operation during the verification is part of the trusted amalgamation. Thus, $\widehat{\text{MAC}}_F = (\text{Am}, \widehat{F})$, where \widehat{F} is a subversion-resilient PRF, and Am calls the PRF for tagging and for verification it recomputes the MAC using the implementation of the PRF and compares the result with its input. Thus, the watchdog runs the watchdog of the subversion-resilient PRF. Finally, we can conclude that the PRF presented in Sect. 3.1 (built from a weak PRF) is a subversion-resilient deterministic MAC as well.

Theorem 5. *If \widehat{F} is pseudorandom under subversion, then $\widehat{\text{MAC}}_F = (\text{Am}, \widehat{F})$ is SUF-CMA-secure under subversion assuming a trusted $=$ operation.*

Corollary 1. *The specification $\widehat{\text{MAC}}_{F^{(\ell)}} = (\text{Am}, \widehat{F}^{(\ell)})$ is SUF-CMA-secure under subversion assuming a trusted $=$ operation.*

5 Symmetric Encryption

In this section, we construct subversion-resilient symmetric encryption using a classical construction based on PRFs that has indistinguishable encryptions under a chosen-message attack. A symmetric encryption scheme works on a keyspace \mathcal{K} , message space \mathcal{M} , and ciphertext space \mathcal{C} . As usual, we have $\mathcal{K} = \bigcup_{\lambda \in \mathbb{N}} \mathcal{K}_\lambda$, $\mathcal{M} = \mathcal{M}_{\lambda \in \mathbb{N}}$, and $\mathcal{C} = \bigcup_{\lambda \in \mathbb{N}} \mathcal{C}_\lambda$.

$\text{Exp}_{\mathcal{A}, \text{SE}}^{\text{IND}\$-\text{CPA}}(1^\lambda)$ <hr style="border: 0; border-top: 1px solid black; margin: 5px 0;"/> $1 : K \xleftarrow{\$} \text{KGen}(1^\lambda); b \xleftarrow{\$} \{0, 1\}$ $2 : b' \xleftarrow{\$} \mathcal{A}^{\text{RoR}_b(\cdot)}(1^\lambda)$ $3 : \text{return } (b = b')$
--

Fig. 5. Security experiment for IND $\$$ -CPA-security, where $\text{RoR}_0(M) = \$_K(M)$ and $\text{RoR}_1(M) = \text{Enc}(K, M)$ such that $\$_K(M)$ computes $C \xleftarrow{\$} \text{Enc}(K, M)$ and if $C = \perp$, outputs \perp , and otherwise, outputs a random string of length $|C|$.

Definition 5. We call a triple $\text{SE} = (\text{KGen}, \text{Enc}, \text{Dec})$ a symmetric encryption scheme SE with key space \mathcal{K} , message space \mathcal{M} , and ciphertext space \mathcal{C} . The randomized key generation algorithm KGen outputs upon the security parameter 1^λ as input a key $K \xleftarrow{\$} \mathcal{K}_\lambda$. The randomized encryption algorithm Enc is given a key $K \in \mathcal{K}_\lambda$ and a message $M \in \mathcal{M}_\lambda$ and returns either a ciphertext $C \in \mathcal{C}_\lambda$ or a symbol \perp . The deterministic decryption algorithm Dec is given a key K and a ciphertext C , and returns either a message $M \in \mathcal{M}_\lambda$ or the symbol \perp .

We say that SE has perfect correctness, i.e., for all $K \in \mathcal{K}_\lambda$, all $M \in \mathcal{M}_\lambda$ and all $C \in \text{Supp}(\text{Enc}(K, M))$, we have $\text{Dec}(K, C) = M$ if $C \neq \perp$. For security, we consider IND $\$$ -CPA-security (i.e., indistinguishability from random bits) [35, 36], which can be shown to imply IND-CPA-security in the left-or-right sense (see, e.g., [6]) by a straightforward reduction.

Definition 6. Let SE be a symmetric encryption scheme and let $\text{Exp}_{\mathcal{A}, \text{SE}}^{\text{IND}\$-\text{CPA}}(1^\lambda)$ be defined as shown in Fig. 5. We define $\text{Adv}_{\mathcal{A}, \text{SE}}^{\text{IND}\$-\text{CPA}}(1^\lambda) := \Pr[\text{Exp}_{\mathcal{A}, \text{SE}}^{\text{IND}\$-\text{CPA}}(1^\lambda) = 1]$ and say that SE is IND $\$$ -CPA-secure if $\text{Adv}_{\text{SE}}^{\text{IND}\$-\text{CPA}}(\mathcal{A})$ is negligible for all ppt adversaries \mathcal{A} .

Symmetric Encryption from PRFs. In Sect. 3, we construct subversion-resilient PRFs. A classical use case of PRFs is the construction of symmetric encryption. Recall the following construction of a stream cipher $\text{SE}_{\text{KS}} = (\text{KGen}_{\text{KS}}, \text{Enc}_{\text{KS}}, \text{Dec}_{\text{KS}})$ based on a PRF $\text{KS}: \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$. KGen on input 1^λ outputs a uniform key $K \xleftarrow{\$} \mathcal{K}_\lambda$, Enc_{KS} on input a key $K \in \mathcal{K}_\lambda$ and a message $M \in \mathcal{R}_\lambda$, outputs a ciphertext (IV, C) , where $\text{IV} \xleftarrow{\$} \mathcal{D}$ and $C := \text{KS}(K, \text{IV}) \oplus M$, and Dec_{KS} on input a key $K \in \mathcal{K}_\lambda$ and a ciphertext $(\text{IV}, C) \in \mathcal{D}_\lambda \times \mathcal{R}_\lambda$, outputs $M := \text{KS}(K, \text{IV}) \oplus C$.

Subversion-Resilience of Stream Ciphers. Next, we show that the above construction is subversion-resilient if the underlying function KS is a subversion-resilient weak PRF. Thus, we consider $\widehat{\text{SE}}_{\text{KS}} = (\text{Am}, \widehat{\text{KS}})$ where $\widehat{\text{KS}}$ is the specification of a subversion-resilient weak PRF and Am randomly chooses IVs and then calls the underlying PRF and applies the trusted \oplus operation.

Theorem 6. *If $\widehat{\text{KS}}$ is weakly pseudorandom under subversion, then $\widehat{\text{SE}}_{\text{KS}}$ is IND\$-CPA-secure under subversion given that the randomness generation and \oplus operation are trusted.*

Proof (Sketch). The watchdog runs the watchdog for $\widehat{\text{KS}}$. The main idea of the proof is that the output of KS is indistinguishable from uniformly random bits for uniformly random IVs and any adversary, even under subversion. Thus for any message M , the output of Enc is indistinguishable from uniformly random bits under subversion for any adversary as well. Hence, $\widehat{\text{SE}}_{\text{KS}}$ is IND\$-CPA-secure under subversion.

Key Stream Derivation of CTR. It remains to demonstrate that this construction can actually be instantiated. A popular instantiation of the above construction is the (randomized) counter mode (CTR\$). The above construction in combination with KS_{CTR} defined next yields CTR\$. Given a PRF $F: \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$, we define the function $\text{KS}_{\text{CTR}}: \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}^\ell$ as

$$(K, \text{IV}) \mapsto (F(K, \text{IV}), F(K, \text{IV} \oplus \langle 1 \rangle_n), \dots, F(K, \text{IV} \oplus \langle \ell - 1 \rangle_n)),$$

where $\ell \in \text{poly}(\lambda)$ and $\langle i \rangle_n$ denotes the n -bit binary representation of $i \in \mathbb{N}$. Next, we show that KS_{CTR} is a subversion-resilient weak PRF assuming that handling the state (i.e. the counter) is modeled as part of the amalgamation.

Theorem 7. *If \widehat{F} is pseudorandom under subversion, then $\widehat{\text{KS}}_{\text{CTR}}$ is a subversion-resilient weak PRF under the assumption that randomness generation and the \oplus operation are trusted.*

Proof (Sketch). The watchdog for $\widehat{\text{KS}}_{\text{CTR}}$ runs the watchdog for \widehat{F} as a subroutine. To prove that the KS_{CTR} is secure even if the building block F is subverted, the main idea is as follows: If F is indistinguishable from random (even under subversion), then KS_{CTR} is indistinguishable from random for uniformly random inputs as long as the sequence $(\text{IV}, \dots, \text{IV} \oplus \langle \ell - 1 \rangle_n)$ does not overlap for two PRF queries. This is because an adversary directly could observe the structure and distinguish the function from random. By a simple argument, one can bound this probability by $\frac{q^2 \ell}{|\mathcal{D}|}$, which is negligible for polynomial block length ℓ , polynomial IV length $\log(|\mathcal{D}|)$, and a polynomial number of PRF queries q .

Note that KS_{CTR} is not a PRF as the adversary can simply choose the IVs such that they overlap which enables the adversary to distinguish KS_{CTR} from a truly random function with overwhelming probability. Finally, Theorem 6 and 7 imply that the randomized counter mode CTR\$ is subversion-resilient.

Corollary 2. *Let CTR\$ be the stream cipher construction above instantiated with KS_{CTR} . Then, $\widetilde{\text{CTR}}\$$ is IND\$-CPA-secure under subversion given that the randomness generation and the \oplus operation are trusted.*

Thus, IND\$-CPA security directly follows from the security of the underlying PRF. While security is preserved, this does not automatically mean that correctness is preserved also: the decryption algorithm is *not* executed in the IND\$-CPA security experiment, but is fundamental for correctness. As discussed by Russell, Tang, Yung, and Zhou [38] this would allow for censorship of chosen messages. If we would consider a black box decryption algorithm, perfect correctness is impossible to achieve, as a single input trigger (for example for C^* the decryption always output a constant value) violates the perfect correctness requirement while highly unlikely to being detected by a watchdog. Nevertheless, as in our construction the adversary only provides an implementation of the underlying PRF, we see that our construction automatically satisfies *perfect* correctness.

Theorem 8. *The specification $\widehat{\text{SE}}_{\text{KS}}$ is perfectly correct.*

Proof. Correctness follow from the “canonical decryption”⁵ of the stream cipher as the same value as during the encryption procedure are computed. Thus, even if $\text{KS}(K, IV)$ deviates from the specification, the subverted output cancels out by the \oplus operation: $\widetilde{\text{Dec}}(K, \widetilde{\text{Enc}}(K, M)) = \widetilde{\text{KS}}(K, IV) \oplus \widetilde{\text{KS}}(K, IV) \oplus M = M$.

Note that previous works [38] also achieved correctness, but tolerated a negligible decryption error. This is because the authors viewed the decryption algorithm as an algorithm with a public input distribution and can check consistency with the specification up to a negligible failure probability. Due to more fine grained access to the decryption procedure, we can guarantee *perfect* correctness.

6 Authenticated Encryption

We now see that the classical Encrypt-then-MAC approach grants us subversion-resilient authenticated encryption, given subversion-resilient building blocks. An authenticated encryption scheme works on a keyspace \mathcal{K} , message space \mathcal{M} , data space \mathcal{D} , and ciphertext space \mathcal{C} . As usual, we have $\mathcal{K} = \bigcup_{\lambda \in \mathbb{N}} \mathcal{K}_\lambda$, $\mathcal{M} = \mathcal{M}_{\lambda \in \mathbb{N}}$, $\mathcal{D} = \bigcup_{\lambda \in \mathbb{N}} \mathcal{D}_\lambda$, and $\mathcal{C} = \bigcup_{\lambda \in \mathbb{N}} \mathcal{C}_\lambda$. In the following, we assume that the message space \mathcal{M}_λ and the ciphertext space \mathcal{C}_λ contain a special symbol \perp .

Definition 7. *We call a triple $\text{AD} = (\text{KGen}, \text{Enc}, \text{Dec})$ a symmetric encryption scheme with associated data for key space \mathcal{K} , message space \mathcal{M} , data space \mathcal{D} , and ciphertext space \mathcal{C} . The randomized key generation algorithm KGen outputs a key $K \stackrel{\$}{\leftarrow} \mathcal{K}_\lambda$ upon input the security parameter 1^λ . The randomized encryption algorithm Enc is given a key $K \in \mathcal{K}_\lambda$, a message $M \in \mathcal{M}_\lambda$, associated data $D \in \mathcal{D}_\lambda$ and returns a ciphertext $C \in \mathcal{C}_\lambda$. The deterministic decryption algorithm Dec is given a key $K \in \mathcal{K}_\lambda$, a ciphertext $C \in \mathcal{C}_\lambda$, associated data $D \in \mathcal{D}_\lambda$ and returns a message $M \in \mathcal{M}_\lambda$.*

⁵ By this we mean recomputing a value and applying it via \oplus to the ciphertext in order to decrypt.

$\text{Exp}_{\mathcal{A}, \text{AD}}^{\text{AE}}(1^\lambda)$ <hr style="border: 0.5px solid black;"/> <pre style="margin: 0; padding: 0;"> 1: $Q \leftarrow \emptyset; b \xleftarrow{\\$} \{0, 1\}; K \xleftarrow{\\$} \text{KGen}(1^\lambda)$ 2: if $b == 1$ then 3: $b' \xleftarrow{\\$} \mathcal{A}^{\text{Enc}(K, \cdot, \cdot), \text{Dec}(K, \cdot, \cdot)}(1^\lambda)$ 4: else 5: $b' \xleftarrow{\\$} \mathcal{A}^{\\$K(\cdot, \cdot), \perp(\cdot, \cdot)}(1^\lambda)$ 6: return $b' == b$ </pre>	$\text{Oracle } \$K(M, D)$ <hr style="border: 0.5px solid black;"/> <pre style="margin: 0; padding: 0;"> 1: $C \xleftarrow{\\$} \{0, 1\}^{ C }$ 2: $Q = Q \cup \{(M, D), C\}$ 3: return C </pre> <hr style="border: 0.5px solid black;"/> $\text{Oracle } \perp(M, D)$ <hr style="border: 0.5px solid black;"/> <pre style="margin: 0; padding: 0;"> 1: if $((M, D), C') \in Q$ then 2: return (M, D) 3: else return \perp </pre>
---	--

Fig. 6. The security experiment for AEAD. The oracle $\text{Enc}(K, \cdot, \cdot)$ expects for $K \in \mathcal{K}_\lambda$ a message $M \in \mathcal{M}_\lambda$ and data $D \in \mathcal{D}_\lambda$, while $\text{Dec}(K, \cdot, \cdot)$ expects a ciphertext $C \in \mathcal{C}_\lambda$ and data $D \in \mathcal{D}_\lambda$.

A symmetric encryption scheme with associated data $\text{AD} = (\text{KGen}, \text{Enc}, \text{Dec})$ is said to be perfectly correct if for all $K \in \mathcal{K}_\lambda$, $M \in \mathcal{M}_\lambda$, and $D \in \mathcal{D}_\lambda$ it holds $\text{Dec}(K, \text{Enc}(K, (M, D)), D) = (M, D)$.

Definition 8. Let AD be a symmetric encryption scheme with associated data and let $\text{Exp}_{\mathcal{A}, \text{AD}}^{\text{AE}}(1^\lambda) = 1$ be defined as shown in Fig. 6. We define $\text{Adv}_{\mathcal{A}, \text{AD}}^{\text{AE}}(1^\lambda) := \Pr[\text{Exp}_{\mathcal{A}, \text{AD}}^{\text{AE}}(1^\lambda) = 1]$ and say that AD is AE-secure if $\text{Adv}_{\mathcal{A}, \text{AD}}^{\text{AE}}(1^\lambda)$ is negligible for all ppt adversaries \mathcal{A} .

Usually, the experiment requires that the adversary does not ask for decryption of outputs of the encryption oracle. For $b = 0$ the experiment cannot output a message, since its just given a random string. As this would trivially break security (and the adversary already knows the answer to the query), this case is excluded in most works. Hence, there is no need to manage the set Q explicitly. In the context of subversion this approach unfortunately also rules out a natural, challenging attack. An adversary could provide an implementation for the decryption algorithm, which instead of a certain message M^* simply outputs the secret key, allowing the adversary to trivially break security. If the decryption algorithm is modeled as a black box, this attack seems unavoidable since a watchdog cannot efficiently detect the trigger message M^* . Even amalgamation of several subverted components cannot prevent this attack if no trusted component is ever used, as an input trigger for the “first” component can again directly lead to input trigger of the next subverted component and so on. Thus, some sort of trusted operation is necessary in order to defend against these kind of attacks. As seen by our construction of stream ciphers with “canonical decryption” in combination with a trusted XOR operation, perfect correctness is guaranteed. In order to include this attack in our model, we change the behavior of the oracles in the experiment for $b = 0$ by introducing book keeping of the queries. This allows the adversary to ask for decryptions of encryption queries without trivially breaking security. An interesting side effect of this definition is that every scheme satisfying this security notion also *needs* to satisfy correctness

(although only up to negligible failure probability), as the adversary would be able to distinguish the two worlds of the experiment otherwise.

6.1 Achieving Subversion-Resilience via Encrypt-then-MAC

The classical way to construct authenticated encryption relies on the use of a MAC that is applied *after* encryption. Decryption then first verifies the MAC and afterwards performs the underlying decryption algorithm. Due to the strong unforgeability of the MAC, an adversary cannot make use of decryption and the security experiment reduces to IND $\$$ -CPA-security. This is useful for achieving subversion-resilience, as a symmetric encryption which is IND $\$$ -CPA secure under subversion does not give any security guarantees for the decryption algorithm and avoids input trigger attacks. While our construction of stream ciphers guarantees correctness under random inputs, this cannot be guaranteed if the adversary can freely choose the inputs for the decryption algorithm. The reason for this again are input trigger attacks, since no watchdog knows the distribution of the queries made by the adversary. The Encrypt-then-MAC approach with subversion-resilient MAC ensures that input trigger attacks are ruled out even if the verify algorithm is subverted. In combination with a subversion-resilient encryption scheme the adversary needs forge a MAC to obtain a decryption of a ciphertext that it did not obtain as an output of the encrypt oracle.

Encrypt-then-MAC. Let $\text{MAC} = (\text{Gen}, \text{Tag}, \text{Vf})$ be a MAC and $\text{SE}_{\S} = (\text{KGen}_{\S}, \text{Enc}_{\S}, \text{Dec}_{\S})$ be a symmetric encryption scheme. Then, the symmetric encryption scheme with associated data $\text{AD} = \text{AD}_{\text{SE}_{\S}, \text{MAC}} = (\text{KGen}_{\text{AD}}, \text{Enc}_{\text{AD}}, \text{Dec}_{\text{AD}})$ is defined as follows: The key generation algorithm $\text{KGen}_{\text{AD}}(1^{\lambda})$ first obtains a key $K_{\text{MAC}} \xleftarrow{\S} \text{KGen}_{\text{MAC}}(1^{\lambda})$, another key $K_{\S} \xleftarrow{\S} \text{KGen}_{\S}(1^{\lambda})$ and outputs $K = (K_{\text{MAC}}, K_{\S})$. The encryption algorithm $\text{Enc}_{\text{AD}}(K = (K_{\text{MAC}}, K_{\S}), M, D)$ first calls the underlying encryption algorithm Enc_{\S} to compute a ciphertext $C_{\S} \xleftarrow{\S} \text{Enc}_{\S}(K_{\S}, M)$, then produces a tag $T \xleftarrow{\S} \text{Tag}(K_{\text{MAC}}, C_{\S} \parallel D)$, and outputs $C = (C_{\S}, T)$. The decryption algorithm $\text{Dec}_{\text{AD}}(K = (K_{\text{MAC}}, K_{\S}), C = (C_{\S}, T), D)$ first verifies the MAC of C by computing $b \xleftarrow{\S} \text{Vf}(K_{\text{MAC}}, C_{\S} \parallel D, T)$. If $b = 0$ (i.e. the verification failed), it returns \perp . Else, it computes $M \xleftarrow{\S} \text{Dec}_{\S}(K_{\S}, C_{\S})$ and returns M . The straightforward reduction to the security of MAC and SE_{\S} can be used to obtain the following well-known theorem.

Theorem 9 ([9]). *If MAC is SUF-CMA-secure and SE_{\S} is IND $\$$ -CPA-secure, then AD is AE-secure.*

Correctness of SE is directly inherited by AD as the MAC does not change the correctness of the underlying encryption scheme.

Theorem 10. *If $\widehat{\text{SE}}_{\S}$ is perfectly correct, then $\widehat{\text{AD}}$ is perfectly correct.*

We prove that the Encrypt-then-MAC approach is indeed sound if both the encryption scheme as well as the MAC are subversion-resilient, following the proof idea of Bellare and Namprempre [8, 9].

Theorem 11. *If $\widehat{\text{SE}}$ is IND $\text{\$}$ -CPA-secure under subversion, $\widehat{\text{MAC}}$ is SUF-CMA-secure under subversion and $\widehat{\text{SE}}$ is correct, then $\widehat{\text{AD}}$ is AE-secure under subversion and correct.*

Proof (Sketch). The watchdog for $\widehat{\text{AD}}$ runs the watchdog for $\widehat{\text{SE}}$ and $\widehat{\text{MAC}}$. Note that there is no need for testing the components in combination, as both allow arbitrary input distributions while being subverted and are executed independently. Assuming that the watchdog does not detect subversion, we can follow the proof of Bellare and Namprepre [8,9] for the Encrypt-then-MAC approach. The main difference is to base the security on the subversion-resilience of the building blocks rather than the “classical” security properties. However, these only differ by the preceded check by the watchdog and that usage of the implementation instead of the specifications. Thus, in the same way IND $\text{\$}$ -CPA security in the classical setting does grant any security guarantees for the decryption algorithm, we also do not need to immunize the subverted decryption algorithm. First, we replace the decryption algorithm for $b = 1$ with an oracle that always answers with the \perp symbol, except if the input was obtained by querying the encrypt oracle. If the output of the encrypt oracle was handed to the decryption oracle, the adversary always obtains correct answer, either by the correctness of the encryption scheme for $b = 1$ or due to the book keeping if $b = 0$. Now assume C was not obtained via the encrypt oracle. In case C is malformed and the decryption oracle returns a \perp symbol, the adversary cannot distinguish this from an oracle which always returns the \perp symbol. The last case that remains is that C is a valid ciphertext and such that the encrypted message (M, D) was *not* issued to the encryption oracle. This case is again not possible, since any adversary reaching this case would have successfully forged a tag for (M, D) , thus breaking subversion resilience of $\widehat{\text{MAC}}$. Therefore we can “disable” the decryption oracle. Then however, we can base security entirely on the subversion-resilience of $\widehat{\text{SE}}$, as the adversary can only make effective use of the encrypt oracle. We replace the encrypt oracle with an oracle returning random bits. This change is again indistinguishable by the subversion-resilience of $\widehat{\text{SE}}$. We changed all oracles so that for both choices of b the adversary is given access to the same oracles, thus being unable to win the experiment apart from guessing the bit b .

Acknowledgements. Supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme, grant agreement 802823.

References

1. Armour, M., Poettering, B.: Substitution attacks against message authentication. IACR Trans. Symm. Cryptol. **2019**(3), 152–168 (2019). <https://doi.org/10.13154/tosc.v2019.i3.152-168>
2. Armour, M., Poettering, B.: Subverting decryption in AEAD. In: Albrecht, M. (ed.) IMACC 2019. LNCS, vol. 11929, pp. 22–41. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35199-1_2

3. Ateniese, G., Francati, D., Magri, B., Venturi, D.: Public immunization against complete subversion without random oracles. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 2019. LNCS, vol. 11464, pp. 465–485. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21568-2_23
4. Ateniese, G., Magri, B., Venturi, D.: Subversion-resilient signature schemes. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 364–375. ACM Press, October 2015. <https://doi.org/10.1145/2810103.2813635>
5. Bauer, B., Farshim, P., Mazaheri, S.: Combiners for backdoored random oracles. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 272–302. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_10
6. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th FOCS, pp. 394–403. IEEE Computer Society Press, October 1997. <https://doi.org/10.1109/SFCS.1997.646128>
7. Bellare, M., Jaeger, J., Kane, D.: Mass-surveillance without the state: strongly undetectable algorithm-substitution attacks. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 1431–1440. ACM Press, October 2015. <https://doi.org/10.1145/2810103.2813681>
8. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_41
9. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. *J. Cryptol.* **21**(4), 469–491 (2008). <https://doi.org/10.1007/s00145-008-9026-x>
10. Bellare, M., Paterson, K.G., Rogaway, P.: Security of symmetric encryption against mass surveillance. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 1–19. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_1
11. Bemmam, P., Chen, R., Jager, T.: Subversion-resilient public key encryption with practical watchdogs. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12710, pp. 627–658. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75245-3_23
12. Berndt, S., Liskiewicz, M.: Algorithm substitution attacks from a steganographic perspective. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 1649–1660. ACM Press, October/November 2017. <https://doi.org/10.1145/3133956.3133981>
13. Bhattacharyya, R., Nandi, M., Raychaudhuri, A.: Crooked indifferentiability of enveloped XOR revisited. In: Adhikari, A., Küsters, R., Preneel, B. (eds.) INDOCRYPT 2021. LNCS, vol. 13143, pp. 73–92. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92518-5_4
14. Bogdanov, A., Rosen, A.: Pseudorandom functions: three decades later. In: *Tutorials on the Foundations of Cryptography*. ISC, pp. 79–158. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57048-8_3
15. Bossuat, A., Bultel, X., Fouque, P.-A., Onete, C., van der Merwe, T.: Designing reverse firewalls for the real world. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12308, pp. 193–213. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58951-6_10
16. Brown, D.R.L., Gjøsteen, K.: A security analysis of the NIST SP 800-90 elliptic curve random number generator. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 466–481. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_26

17. Chakraborty, S., Dziembowski, S., Nielsen, J.B.: Reverse firewalls for actively secure MPCs. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12171, pp. 732–762. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_26
18. Chakraborty, S., Magri, B., Nielsen, J.B., Venturi, D.: Universally composable subversion-resilient cryptography. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part I. LNCS, vol. 13275, pp. 272–302. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-06944-4_10
19. Checkoway, S., et al.: A systematic analysis of the juniper dual EC incident. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 468–479. ACM Press, October 2016. <https://doi.org/10.1145/2976749.2978395>
20. Chen, R., Mu, Y., Yang, G., Susilo, W., Guo, F., Zhang, M.: Cryptographic reverse firewall via malleable smooth projective hash functions. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 844–876. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_31
21. Chow, S.S.M., Russell, A., Tang, Q., Yung, M., Zhao, Y., Zhou, H.-S.: Let a non-barking watchdog bite: cliptographic signatures with an offline watchdog. In: Lin, D., Sako, K. (eds.) PKC 2019. LNCS, vol. 11442, pp. 221–251. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17253-4_8
22. Degabriele, J.P., Farshim, P., Poettering, B.: A more cautious approach to security against mass surveillance. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 579–598. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48116-5_28
23. Dodis, Y., Farshim, P., Mazaheri, S., Tessaro, S.: Towards defeating backdoored random oracles: indifferentiability with bounded adaptivity. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12552, pp. 241–273. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64381-2_9
24. Dodis, Y., Ganes, C., Golovnev, A., Juels, A., Ristenpart, T.: A formal treatment of backdoored pseudorandom generators. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 101–126. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_5
25. Dodis, Y., Mironov, I., Stephens-Davidowitz, N.: Message transmission with reverse firewalls—secure communication on corrupted machines. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 341–372. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_13
26. Fischlin, M., Janson, C., Mazaheri, S.: Backdoored hash functions: immunizing HMAC and HKDF. In: Chong, S., Delaune, S. (eds.) CSF 2018 Computer Security Foundations Symposium, pp. 105–118. IEEE Computer Society Press (2018). <https://doi.org/10.1109/CSF.2018.00015>
27. Fischlin, M., Mazaheri, S.: Self-guarding cryptographic protocols against algorithm substitution attacks. In: Chong, S., Delaune, S. (eds.) CSF 2018 Computer Security Foundations Symposium, pp. 76–90. IEEE Computer Society Press (2018). <https://doi.org/10.1109/CSF.2018.00013>
28. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: 25th FOCS, pp. 464–479. IEEE Computer Society Press, October 1984. <https://doi.org/10.1109/SFCS.1984.715949>
29. Goldreich, O., Goldwasser, S., Micali, S.: On the cryptographic applications of random functions (extended abstract). In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 276–288. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_22

30. Maurer, U., Sjödin, J.: A fast and key-efficient reduction of chosen-ciphertext to known-plaintext security. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 498–516. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_29
31. Maurer, U., Tessaro, S.: Basing PRFs on constant-query weak PRFs: minimizing assumptions for efficient symmetric cryptography. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 161–178. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_11
32. Mironov, I., Stephens-Davidowitz, N.: Cryptographic reverse firewalls. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 657–686. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_22
33. Naor, M., Reingold, O.: Synthesizers and their application to the parallel construction of pseudo-random functions. *J. Comput. Syst. Sci.* **58**(2), 336–375 (1999)
34. Perlroth, N., Larson, J., Shane, S.: Secret documents reveal NSA campaign against encryption (2013). <https://archive.nytimes.com/www.nytimes.com/interactive/2013/09/05/us/documents-reveal-nsa-campaign-against-encryption.html>
35. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) ACM CCS 2002, pp. 98–107. ACM Press, November 2002. <https://doi.org/10.1145/586110.586125>
36. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Reiter, M.K., Samarati, P. (eds.) ACM CCS 2001, pp. 196–205. ACM Press, November 2001. <https://doi.org/10.1145/501983.502011>
37. Russell, A., Tang, Yung, M., Zhou, H.-S.: Cliptography: clipping the power of kleptographic attacks. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 34–64. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_2
38. Russell, A., Tang, Q., Yung, M., Zhou, H.S.: Generic semantic security against a kleptographic adversary. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 907–922. ACM Press, October/November 2017. <https://doi.org/10.1145/3133956.3133993>
39. Russell, A., Tang, Q., Yung, M., Zhou, H.-S.: Correcting subverted random oracles. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 241–271. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_9
40. Shumow, D., Ferguson, N.: On the possibility of a back door in the NIST sp800-90 dual EC PRNG (2007). <http://rump2007.cr.yp.to/15-shumow.pdf>, cRYPTO 2007 Rump Session
41. Wegman, M.N., Carter, L.: New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.* **22**(3), 265–279 (1981). [https://doi.org/10.1016/0022-0000\(81\)90033-7](https://doi.org/10.1016/0022-0000(81)90033-7)
42. Young, A., Yung, M.: The dark side of “black-box” cryptography or: should we trust capstone? In: Kobitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 89–103. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_8



Scored Anonymous Credentials

Sherman S. M. Chow¹✉ , Jack P. K. Ma¹, and Tsz Hon Yuen² 

¹ Department of Information Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

{smchow,mpk016}@ie.cuhk.edu.hk

² Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong

thyuen@cs.hku.hk

Abstract. Securely maintaining “credits” of users judging their behavior in past authenticated sessions is vital to encourage user participation, but doing it over anonymous credentials is non-trivial, especially when users would avoid claiming negative credit and escape from blocklisting. Prevalent designs impose an authentication cost linear in the blocklist size or a stringent requirement of *sequential and timely* judgment of each session without retrospective adjustment, as a single unjudged session curbs the authentication of *all* users. We propose scored anonymous credentials, a new design storing a number of *active* sessions with volatile scores downgradable before finalized. Sessions can be judged in any order and at varying times without affecting all users. Any backlog of unjudged sessions only affects the users behind them. We achieve efficiency and flexibility using verifiable shuffle, which is hardly used in existing anonymous blocklisting/reputation enforcement systems.

1 Introduction

Anonymity has always been an important issue on the Internet. In the early days, most people (wrongly) assumed anonymity on the Internet by using pseudonyms, but the service provider (SP) might use IP addresses to identify the users. Anonymity networks [29, 36] were then developed to make tracing IP addresses more difficult. However, this may not be directly applicable to services that require users to register and authenticate. Using the same registered pseudonym to access the service each time links user behavior across sessions. It is possible to uniquely profile a user across different services by collecting this information [41].

Anonymous credentials [18] ensure unlinkable authentications. However, users may “misbehave” in some sessions exploiting their anonymity. To illustrate, Wikipedia users (contributors/reviewers) may prefer their identities to be

Sherman Chow is supported in part by the General Research Funds (CUHK 14210621 and 14209918), University Grants Committee, Hong Kong, and is also grateful for their recognition in granting the Early Career Award on “Accountable Privacy in Online Communication” (CUHK 439713). We thank Kin-Ying Yu for inspiration from his Ph.D. thesis, and Russell Lai and reviewers for their helpful comments.

masked so that their views would not represent any belonging organizations or communities (e.g., a company/political party) and avoid their online behavior causing any consequence in their “real” life, especially in places with restricted freedom of speech. On the other hand, Wikipedia has its own set of “subjective” policies and guidelines (e.g., concerning copyright) that users must follow. Users who violate such policies repetitively and deliberately should be penalized.

It is challenging to support *subjective* revocation (cf., legal contract) in anonymous authentication without relying on any *trusted third party* (TTP). Some systems only support *objective* revocation based on mathematically or algorithmically evaluable claims of misbehavior (cf., smart contract), such as double spending in e-cash [23,37] or double voting [24]. Revocable anonymous credentials [2,3,17,19,28] often assume users to be revoked had been deanonymized by some external means. TTP-based approaches, such as group/traceable signatures, require a revocation authority to use a non-public database gathered when the user first joins the system [7,22] or a secret key to recover the signer identity [7,26] or create a tracing token [1,48]. The TTP is assumed not to violate user privacy unnecessarily. Simply, “off-the-shelf” credentials themselves do not support TTP-free subjective revocation [44].

Generalizing revocation, reputation can be used as a basis for granting privileges to special services or imposing limitations on users based on their behaviors. It is important to encourage participation, especially when participation may potentially incur a revocation or real-life consequences. A large-scale online anonymous community has already been developed. For example, Bernstein *et al.* [10] showed that over 90% of posts are anonymous on 4chan.org, which has a million posts per day¹ and 22+ million unique monthly users. However, registration is not needed for posting anonymously in these forums currently. So, reputation cannot be maintained. We refer to [35] for a survey on reputation management.

Updatable anonymous credential is useful for adding reputation. Each user is associated with a *reputation score*, an aggregated sum of all individual reputations decided by the SP (possibly after hearing from other users) one gains from each contribution. A high reputation could mean the user has successfully helped many others and can be considered more trustworthy. Reputation can also somewhat mitigate *sybil attacks* since an attacker would need to give up any previously earned reputation for registering anew.

1.1 A Critical Review of Existing Systems

Existing (updatable) anonymous credential systems with subjective revocation, however, are designed for applications with lower traffic and anonymity needs (e.g., assuming anonymity is needed for 20% of all Wikipedia posts and is meant to handle about 100,000 anonymous posts per day [4,5]). They are not suitable for a large-scale community with millions of posts per day, e.g., Reddit.

¹ Statistic at <https://4stats.io>.

All works reviewed below assume one SP serving many clients. There is no separate (trusted) entity for revocation. For simplicity, this line of works assumes the SP is both the credential issuer and verifier, and has the final say in the score for each session, which can be published on a public bulletin board.

After a successful user authentication, the SP will assign a publicly-known session identifier. The user credential will be attached to a ticket associated to the session identifier. These two terms are often used interchangeably in this paper.

Table 1 compares known systems in terms of their desired properties.

Blocklisting/Reputation with Linear (Proving) Cost. BLAC [44] and EPID [13] are two early TTP-free systems. Their blocklist consists of “*deterministic tags*” directly available from the authentication transcripts of sessions that should be blocklisted. Revocation is done by proving that the user secret in the credential could not have generated any tag in the blocklist. This takes $O(L)$ time, where L is the blocklist size. Moreover, the blocklist *never shrinks*. It can become very long quickly since malicious users are motivated to keep damaging (e.g., spamming or committing vandalism) in a short period before getting caught. The *whole* system keeps slowing down as times go by, affecting all (honest) users.

BLACR [5] extends BLAC to also support reputation with three lists, \mathbb{L}_+ , \mathbb{L}_- , and \mathbb{L}_B , storing the tags of positively scored, negatively scored, and blocked sessions, respectively. Authentication now runs in $O(|\mathbb{L}_+| + |\mathbb{L}_-| + |\mathbb{L}_B|)$ time, which is even worse than BLAC’s $O(L)$ time complexity, where $L = |\mathbb{L}_B|$.

SNARKBlock [43] improves BLAC with an optimized zkSNARK-based proof protocol that allows reusing proofs against chunks of unchanged blocklist. If a session is later unblocked, all must recompute the proof w.r.t. the affected chunk.

Discouraging Infrequent Users. To tackle the linear complexity, BLACR was extended into BLACR-Express [5] with *checkpoints*. Suppose a user has passed an authentication after the i -th checkpoint in the blocklist; the SP marks the value i on the credential. Any future blocklist checking of this *updated* credential only needs to be done with respect to ΔL entries put into the blocklist after the i -th checkpoint. This discourages infrequent users from logging in since their authentication request before catching up with the latest checkpoint will be slower and stand out from the crowd, degrading their anonymity. Users are motivated to rush to get their express passes, forcing the SP to handle a burst of requests at each checkpoint, which might result in a denial of service.

Stringent Timelines for Blocklisting. PEREA [45] changes the authentication semantic for reducing the $O(L)$ complexity to $O(K)$, where K is the fixed number of *tickets* in a credential. Each authentication *replaces* the oldest of the K tickets with a new one, so misbehavior must be caught within the “revocation window” of K authentications. Since malicious users are motivated to wash off their bad records, it imposes a *stringent requirement* for the SP to evaluate each session *sequentially and timely*. The same applies to FAUST [38] and PERM [4]. In Table 1, PERM is marked to have no score consistency since the washing-out problem makes the credential score inconsistent with the judgment intention.

Table 1. Summary of Related Work: Halt-free systems should have complexity independent on any global list, without halting authentication due to a single hard-to-judge session; SNARKBlock is like BLAC, and PEREA is like PERM in this table.

Scheme	Halt-free	Score Consistency	Rate Limit	Cred. Update	Dynamic Judgment
BLAC	✓	✗	✗	✗	Full Flexibility
BLACR	✓	✓	✗	✗	Non-“Checkpointed”
PERM	✗	✗	✗	✓	Block-then-Forgive
PE(AR) ²	✓	✗	✗	✓	Block-then-Forgive
FARB	✗	✓	✓	✓	Block-then-Forgive
SAC	✓	✓	✓	✓	Before Finalization

Block First, Forgive Later. PERM and PE(AR)² [47], which do not prove w.r.t. the whole global list, only support score upgrades – an easy option as a user is motivated to claim them. Indeed, PE(AR)² does not force redeeming of negative scores by itself. Also, the SP might take strict “block-first-forgive-later” measures.

Unavoidably Halting Innocent Users. PERM [4] takes the revocation-window model further, which forces the session identifiers to increase sequentially. The goal is to remove the use of accumulator in PEREA [45] for reducing user computation (albeit with a slightly higher server-side cost) via using a *global* judgment pointer pointing to the latest authenticated session that has been *judged* (e.g., free from misbehavior). The eldest unjudged session of every credential should not be “too far away” from the global pointer for successful authentication. FARB [46] further exploits the sequential order assumption, which replaces K disjunctive proofs (judged or unjudged) with simply one range proof.

Such an authentication semantic forbids any user from redeeming the judged session in an arbitrary order (or it violates the increasing-identifier invariant). The existence of one hard-to-judge session will affect “innocent” users who have nothing to do with that session to an even greater extent than PEREA (which essentially maintains a local judgment pointer compared by counting).

Moreover, even when the global halting is “lifted,” *i.e.*, that controversial session is eventually finalized, all users are motivated to redeem all redeemable sessions in a rush, similar to the checkpoint design of BLACR-Express [5].

We stress that this is a design oversight of all these systems under the revocation-window model, creating a technical restriction incompatible with operational characteristics. In other words, ideally, the fact that some sessions take longer to judge should only affect their originators but not *all system users*.

1.2 Our Contribution

We propose scored anonymous credentials (SAC) with a number of features.

Avoiding Queue Structure or Disjunctive Proof. All K tickets a user holds are assigned with an initial score, e.g., 0, at their creation time. These tickets are

marked as “active” and can only be removed from the credential when the SP finalizes them. This is in contrast to PERM/PEREA, which always removes the oldest ticket and adds a new one to the credential upon authentication.

To assure the SP of the ticket statuses in a credential, authentication requires proving that each ticket is either an *active* one with a volatile score, a judged one with a *finalized* score, or a dummy ticket. Instead of the apparent need for a disjunctive proof (extensively used in PERM/PEREA), we employ a signature-based set membership proof for each ticket (with its optimization and complexity to be described shortly) against a global ticket list with volatile/finalized scores.

Shuffling and Summing Up All Tickets. The user can redeem (and remove) any finalized or dummy tickets. To highlight a novelty of our new design, we do not use complicated zero-knowledge proof (ZKP) to explicitly force a user to redeem/claim a session that is finalized or with its volatile score downgraded. Instead, we use the same signature-based set membership proof to force the inclusion of all tickets (of any status) via a summation of all (volatile/finalized) scores. To privately prove/update this metadata, we employ verifiable shuffle and dummy tickets so the SP/user can judge/redeem tickets *in an arbitrary order*. To the best of our knowledge, no related systems so far use verifiable shuffle.

Solving the Global-Halting Problem. As a *major innovation*, SAC deviates from the two prevalent designs – reliance on a *global* judgment pointer (e.g., PERM) or *global* checkpoints (e.g., BLACR), while still being efficient. The SP can keep a “problematic” session pending prolonged judgment “active” with a volatile score. A credential is only blocked from further authentication if it has K unjudged sessions. Users who are not involved with the problematic cases just authenticate as usual. This leads to “intelligent” rate-limiting *tracing back to originating users*.

Flexibility of Scoring. Supporting active sessions allows an assignment of volatile scores before finalization, making SAC scoring mechanism highly flexible and free from any chronological order of judgment. The SP could temporarily assign a negative score to the session in question, leading to a more natural and benign authentication semantic in which a user with a higher reputation (many prior positive scores) is less likely to be blocked abruptly due to one potential wrongdoing. Yet, once the wrongdoing is confirmed, the session status can be changed from active to finalized, and the user could still be (permanently) banned. Moreover, our efficient ZKP over the status of *all* sessions in a credential also enables *downgradable* scores, while prior works might support only block-then-forgive.

Flexible Anonymity-Efficiency Trade-Off. The use of verifiable shuffle ensures an updated credential remains unlinkable to its old version sharing some old tickets. SAC thus features a *dynamic buffer size*² K . Users with fewer/more than K tickets can pad/remove dummy tickets during each authentication. Frequent users are more willing to choose a larger K , either actually storing many active

² It is non-trivial to reduce the queue size of PEREA/PERM while preserving privacy.

sessions or having some of them being *dummy* sessions of score 0. Less frequent users can opt for a smaller K , which leads to a faster authentication time. This is similar to ring signatures or related techniques [23], in which one can choose which user groups (cf., those using a particular suggested buffer size in SAC) to hide inside. As the SP knows the number of tickets to be removed, the users can agree to remove one (dummy or judged) ticket per authentication for anonymity.

High Efficiency via Optimized Cryptographic Techniques. As explained, a key idea to support redeeming of judged sessions in an *arbitrary order* is to leverage verifiable shuffle, a building block yet to be explored by the research line of TTP-free anonymous credentials with blocklist and reputation. This ensures that the set of sessions is correctly shuffled across the old credential and an updated one. A breakthrough of Bayer and Groth [9] features a “minimal” communication bandwidth of $O(\sqrt{K})$ for shuffling K commitments with almost linear complexity in proving and verification. Recent advances in (succinct) zero-knowledge argument of knowledge (e.g., Bulletproofs [14]) reduce the communication costs to $O(\log K)$ with higher computational costs for the prover and verifier.

Our construction also includes some other optimization for better efficiency. Recall that each authentication ZK-proves that some of its past tickets are in the active ticket list. Like PERM, we do not use pairing-based accumulators [2]; but instead of enforcing a sequential ticket order in a credential to simplify the membership proof, we use a constant-size signature-based set membership proof, albeit without incurring global halting caused by one controversial session.

To claim the score of judged tickets, PERM uses ZKP of signatures, which results in $O(K)$ pairings and exponentiations. We equip our signatures with *an efficient batch verification algorithm accompanied by its zero-knowledge version*. It takes $O(K)$ exponentiation and $O(1)$ pairing only, which greatly improves the efficiency. The use of disjunctive proofs in PERM prevents batching operations.

Finally, we empirically show that SAC outperforms existing related systems, given the simplicity of our design and the cryptographic techniques we employed.

Concurrent Work. SMART [39] extends FARB [46] into a multi-queue design to alleviate the time pressure of finalizing a hard-to-judge session. Each queue keeps tickets that have undergone the same number of transient judgments. A transient judgment takes its effect or is “redeemed” in each authentication, which moves the corresponding ticket from one queue to another. As a credential can be seen as multiple credentials (using the implicit queue design of FARB) bundled together, sequential judging is still enforced per queue but does not affect other queues. This reduces the global-halting effect of sequential judging with a linear cost in the number of queues. However, it requires a more complex ZKP to hide the queue number during redemption. On the other hand, SMART marks the “version” of each transient judgment independently, while enforcing score updates in SAC involves expiring all previously signed judgments and issuing new ones (Sect. 3.5). In Table 1, SMART shares the same characteristics as SAC.

2 Definitions

2.1 Syntax of Scored (or Blocklistable) Anonymous Credentials

We first formulate an abstract definition for anonymous credentials, modeling the two kinds of interactions between *users* and the *service provider* (SP). After the SP sets up the whole system, each user *registers* with the SP anonymously to obtain a credential. A user can then use it to *authenticate* to the SP. For SAC, the SP *updates* the public information to reflect the new score or status assigned to each session according to the action carried out by its user.

We use an existing notation [47] of $2PC(\mathcal{U}(\text{in}_{\mathcal{U}}), \mathcal{S}(\text{in}_{\mathcal{S}})) \rightarrow \{\mathcal{U}(\text{out}_{\mathcal{U}}), \mathcal{S}(\text{out}_{\mathcal{S}})\}$ to represent parties \mathcal{U} and \mathcal{S} interact via the respective \mathcal{U} and \mathcal{S} portion of the 2-party computation protocol 2PC taking in_i and outputting out_i for $i \in \{\mathcal{U}, \mathcal{S}\}$.

Definition 1. *Anonymous credentials involve the algorithms/protocols below.*

$\text{Setup}(1^\lambda) \rightarrow (\text{pp}, \text{sk})$: *The SP inputs in a security parameter λ , and outputs a secret key sk for the SP itself and the public parameter pp .*

$\text{Reg}(\mathcal{U}(\text{pp}), \mathcal{S}(\text{pp}, \text{sk})) \rightarrow \{\mathcal{U}(\text{cred}, \text{A}), \mathcal{S}(\text{pp}')\}$: *The user \mathcal{U} inputs the public parameter pp , while the SP \mathcal{S} takes pp and the secret key sk as input. Upon completion, the user outputs a credential cred and an attribute set A , while the SP outputs an updated public parameter pp' .*

$\text{Auth}(\mathcal{U}(\text{pp}, \text{cred}, \text{A}, f), \mathcal{S}(\text{pp}, \text{sk}, f)) \rightarrow \{\mathcal{U}(b, \text{cred}', \text{A}'), \mathcal{S}(b, \text{pp}')\}$: *The user \mathcal{U} inputs the public parameter pp , its credential cred , its attributes A , and the same access policy f , while the SP \mathcal{S} inputs the public parameter pp , the secret key sk , and an access policy f . Upon completion, \mathcal{U} outputs a bit b indicating whether authentication is successful, an updated credential cred' , and updated attributes A' , while \mathcal{S} outputs the same bit b and an updated public parameter pp' .*

$\text{Update}(\text{pp}, \text{sk}, \text{aux}) \rightarrow \text{pp}'$: *The SP inputs the public parameter pp , the secret key sk , and auxiliary information aux , capturing the score assigned to a certain session as recorded by Auth in pp' . It outputs an updated public parameter pp' .*

Prior Formulations. Earlier works [4, 5, 38, 45] never explicitly give a syntactic framework of blockable anonymous credentials or those also supporting reputation. A notable exception is $\text{PE}(\text{AR})^2$ [47], which features a standalone redemption protocol that updates a credential according to the SP-side update given by two separate algorithms for revocation and scoring (a session) run by the SP.

Our SAC formulation forces the redemption of any negative finalized score (not supported by $\text{PE}(\text{AR})^2$) and hence encapsulates the redemption protocol in Auth , the authentication protocol. For a generic treatment, we also encapsulate all kinds of assignments of score/status into one Update algorithm run by the SP.

In contrast to the different designs of prior works, our definition is generic for an abstract authentication semantic purely based on an attribute set A as a user secret input and an authentication policy f as a common public input.

In what follows, we will also supplement details specific to SAC, e.g., what are in pp , with discussions of alternatives in some prior works.

Credential Attributes. In SAC (and systems like PERM/PEREA), the credential encoding attributes \mathbf{A} stores session identifiers $\mathbb{U} = \{t_1, \dots, t_K\}$ originated from each session (e.g., a forum post or a Wikipedia edit) and score(s) s to be checked against the access policies, e.g., whether they exceed certain thresholds.

Session Judgment. Public session judgments made by the SP are reflected in the updatable public parameter \mathbf{pp} . In SAC, \mathbf{pp} contains a list $\mathbb{L}_A = \{(t_j, s_j, \sigma_j)\}$ of active judgments and a list $\mathbb{L}_J = \{(t_j, \hat{\sigma}_j)\}$ of finalized, judged tickets, where t_j is a ticket unique to each session and s_j is the session score. PERM maintains only \mathbb{L}_A ; while PEREA also maintains \mathbb{L}_J as a list of blocked tickets.

All entries are accompanied by σ_j or $\hat{\sigma}_j$, possibly different forms of signatures issued by the SP. The use of signature varies according to instantiations, which we omit for brevity in parts of our later discussion. Particularly, for SAC, the signatures in \mathbb{L}_J are for set membership proof. We often use a “compact” form $\mathbb{L} = \{(t_i, s_i, \sigma_i, \emptyset/\hat{\sigma}_i)\}$ that combines $\mathbb{L}_{\{A,D,J\}}$. For $t_i \in \mathbb{L}_A \setminus \mathbb{L}_J$, the last entry is \emptyset .

In alternative formulation (e.g., PEREA), \mathbb{L}_J could contain a single (public) cryptographic object, e.g., accumulator, instead of many signatures. \mathbb{L}_A might also be a cryptographic object storing key-value pairs.

Workflow. Below supplements details of SAC calling the (abstract) algorithms:

1. (SAC.Setup) The service provider (SP) setups the credential system.
 - It runs $\text{KGen}(1^\lambda) \rightarrow (\mathbf{pp}, \mathbf{sk})$, where \mathbf{sk} is the secret credential-issuance key.
 - For public parameters \mathbf{pp} , its static part contains the cryptographic parameters and the public (verification) key corresponding to \mathbf{sk} . It also defines default values like the list \mathbb{L}_D of dummy tickets, and the base score s_0 for the initial attributes \mathbf{A}_0 . Its dynamic part contains initially empty lists \mathbb{L}_A and \mathbb{L}_J of active and judged tickets with scores, and a set for recording the nonce revealed by the user during authentication.
2. (SAC.Reg^U \leftrightarrow SAC.Reg^S) A user registers to the SP to obtain a credential.
 - The user and SP run $\text{Reg}(\mathcal{U}(\mathbf{pp}), \mathcal{S}(\mathbf{pp}, \mathbf{sk})) \rightarrow \{\mathcal{U}(\text{cred}, \mathbf{A}_0), \mathcal{S}(\mathbf{pp}')\}$.
 - The initial attributes $\mathbf{A}_0 = (x, q, s_0, \mathbb{U})$ contain user secret x , a nonce q (both kept secret from the SP), an initial score s_0 , and a user ticket set \mathbb{U} , which will be initialized with a set of dummy tickets \mathbb{U}_D (as defined by \mathbf{pp}).
3. (SAC.Auth^U \leftrightarrow SAC.Auth^S) A user who holds a valid credential from SAC.Reg authenticates via $\text{Auth}(\mathcal{U}(\mathbf{pp}, \text{cred}, \mathbf{A}, f), \mathcal{S}(\mathbf{pp}, \mathbf{sk}, f))$, where policy f checks:
 - Nonce q has not been previously used by any authentication.
 - Judged tickets in $\mathbb{U}_J = (\mathbb{U} \cap \mathbb{L}_J)$ are (required³ to be) cleared out.
 - The credential score s plus the score of all tickets in \mathbb{U} satisfies f .
 It outputs \perp if the check fails; otherwise, it updates the credential attributes:
 - Judged or dummy ticket(s) are removed from \mathbb{U} . (Users can redeem any number of dummy tickets, *i.e.*, the set \mathbb{T} containing tickets from $(\mathbb{U}_D \cup \mathbb{U}_J)$.)

³ In PERM/PEREA, the oldest ticket would be removed even if it is not yet judged.

- Nonce q is replaced by a new one (unknown to the SP) chosen by the user.
- Their scores are accumulated to s .
- A new ticket t , chosen by the SP, is appended to the user’s credential.
- Dummy tickets are padded to maintain $|\mathbb{U}| = K$ (if needed).

The SP assigns t with an initial score 0 and updates \mathbb{L}_A with (t, s, σ) applied as an active (and unjudged) session. The SP also records q in \mathbf{pp}' .

4. (SAC.Update) The SP can judge the ticket t with a score s using its key by running $\text{Update}(\mathbf{pp}, \mathbf{sk}, (t, s)) \rightarrow \mathbf{pp}'$ that appends (t, s) (with a signature) to list \mathbb{L}_A . Alternatively, it can also finalize t by adding it to list \mathbb{L}_J . The list of signatures on (t, s) (and t) is made public and is managed by the SP.

2.2 Security Requirements

We consider the requirements of blocklistable anonymous credentials [4, 5, 38, 45]. At a high level, SAC should satisfy the following properties:

- **Completeness.** An honest user can be authenticated by an honest SP if its credential satisfies the access policy, *i.e.*, $f_{\mathbf{pp}}(\mathbf{A}) = 1$.
- **Soundness.** A user must hold a valid credential encoding \mathbf{A} , *i.e.*, $f_{\mathbf{pp}}(\mathbf{A}) = 1$, to authenticate. The updated attribute \mathbf{A}' follows specifications in *Auth*.
- **Anonymity.** The SP can only learn whether an authenticating user satisfies the authentication policy. The SP cannot distinguish the authentication requested by user $i \in \{0, 1\}$ with attributes \mathbf{A}_i if $f_{\mathbf{pp}}(\mathbf{A}_0) = f_{\mathbf{pp}}(\mathbf{A}_1)$.

Soundness. We consider soundness against malicious users similar to and largely relies on the soundness of the underlying ZKP. It covers scoring consistency, *i.e.*, a malicious user cannot claim a ticket owned by others or a wrong score.

In SAC, although a user can choose to “procrastinate” in claiming the score of judged sessions $\mathbb{U} \cap \mathbb{L}_J$, the scores of all tickets in \mathbb{U} are still counted toward the authentication policy. All tickets are initially in an active state, *e.g.*, $(t_i, s_{t_i}, \sigma_{t_i}) \in \mathbb{L}_A$ for all tickets t_i in the system. Scoring consistency requires the score s' computed as a sum of users’ current score s (taking one attribute slot in the credential) and those of the past tickets $\sum_{t \in \mathbb{U}} s_t$ is consistent with \mathbb{L}_A and \mathbb{L}_J .

Anonymity. Anonymity is defined against a passive SP (strictly stronger than eavesdroppers) trying to deanonymize a user who is invoking an authentication instance. Due to the correct functionality, the authentication policy f can distinguish whether (the credential of) a user satisfies the required score threshold. The best anonymity guarantee only holds modulo what is inferrable from f .

Active attacks manipulating f or the score of a session are excluded. For example, the SP, leveraging its role, could put a session of question to the blocklist for identifying if the user being authenticated has originated the now blocked session. Such manipulations are noticeable publicly and leave evidence. A user can refuse to carry out the authentication and possibly complain against the SP.

In more detail, SAC is anonymous if any adversary can only win the anonymity game below with probability negligibly better than a random guess:

1. Adversary \mathcal{A} runs **Setup** and outputs the public parameter pp .
2. \mathcal{A} can instruct a user controlled by challenger \mathcal{C} to run **Reg** with \mathcal{A} . If the protocol outputs a valid credential, \mathcal{C} stores it with a unique user identifier.
3. \mathcal{A} can instruct registered users controlled by \mathcal{C} to run **Auth** over a policy f with \mathcal{A} as the SP. If the user’s credential does not pass f , \mathcal{C} outputs \perp to \mathcal{A} .
4. \mathcal{A} picks two users u_0, u_1 , and sends them to \mathcal{C} .
5. \mathcal{C} checks if both u_0, u_1 pass the policy check f . If so, it flips a coin $b \in \{0, 1\}$ and runs **Auth** using u_b ’s credential, else it outputs \perp .
6. \mathcal{A} wins if it guesses b correctly and \mathcal{C} does not output \perp in Step 5.

One might consider unlinkability, in which the SP cannot tell whether the same user is authenticating. The unlinkability game can be captured by having the adversary pick a user u in Step 4, and the challenger randomly authenticates with a random valid user or u . Intuitively, it is captured by anonymity since the SP only learns if $f_{\text{pp}}(\mathbf{A})$ returns 1 during authentication. This is similar to the equivalence between “left-or-right” and “real-or-random” formulations.

Alternatively, one could formulate a simulation-based definition, requiring the transcript of different **Auth** instances to be uncorrelated from those of the same user or the **Reg** instance. More formally, it asks for a probabilistic polynomial-time simulator that can simulate the view of a corrupted SP in **Auth**. The ideal functionality is in Appendix D.

3 Proposed System

3.1 Building Blocks

Zero-Knowledge Proof-of-Knowledge (ZKPoK). We use a ZKPoK system with correctness, soundness, knowledge extraction, and zero-knowledgeness. It involves a three-move commit-challenge-response Σ -protocol. In the random oracle model, it can be converted into non-interactive signatures/proofs of knowledge.

We use the notation from Camenisch and Stadler [20], like $\text{PoK}\{(\alpha, \rho) : z = g^\alpha h^\rho\}$, to denote such proof of (α, ρ) where $z = g^\alpha h^\rho$ holds. Multiple ZKPoK protocols could be chained into a bigger one for multiple conditions. Compared to ZKPoK, a zero-knowledge argument of knowledge (ZKAoK) is a proof system that satisfies soundness property against any computationally-bounded prover.

Set Membership Proof. Given a commitment $C = g^i h^\rho$ with $g, h \in \mathbb{G}_1$ to a value i and randomness ρ , a set membership proof is a ZKPoK that i belongs to some discrete set Φ . The proof \mathbb{P}_{Set} of Camenisch *et al.* [16] uses the selectively-secure Boneh–Boyen (BB) signature [11]. The SP with the signature key pair $(x, w = f^\beta) \in \mathbb{Z}_p \times \mathbb{G}_2$ alongside with $(g', g'^\beta) \in \mathbb{G}_1^2$ (required by the underlying simulator) publishes signatures $\hat{\sigma}_i = g^{\frac{i}{\beta+i}} \in \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$ on values i in set Φ . The prover picks $r \in_R \mathbb{Z}_p$, and computes $V = \hat{\sigma}_i^r$ and $V' = V^{-i} g^r$. The proof consists of $(V \neq 1_{\mathbb{G}_1}, V')$ and the PoK: $\text{PoK}\{(i, r, \rho) : C = g^i h^\rho; V' = V^{-i} g^r\}$ as follows.

1. The prover picks $s, t, u \in_R \mathbb{Z}_p$ and sends $a = V^{-s}g^t, D = g^s h^u$ to the verifier.
2. The verifier returns a random challenge $c \in_R \mathbb{Z}_p$.
3. The prover sends $z_i = s - ic, z_r = t - rc$, and $z_p = u - \rho c$.
4. The verifier checks if $a = V^{c^i} V^{-z_i} g^{z_r}, D = C^c g^{z_i} h^{z_p}$, and $\hat{e}(V, w) = \hat{e}(V', f)$.

In SAC, we also use BB signature to certify the judged status of a session.

Range Proof. A range proof can be viewed as a special case of set membership proof [15] by defining the set as the range, say, integers in $[A, B]$. This imposes an upper bound value, so we do not need to handle the wrap-around issue in \mathbb{Z}_p even though p is public. This range-proof system is also simple and efficient.

Bulletproofs. Bulletproofs proposed by Bünz *et al.* [14] is a non-interactive zero-knowledge proof protocol without a trusted setup, featuring a proof size only logarithmic in the witness size. Bulletproofs are well suited for proofs for general arithmetic circuits and inner-product relations. Range proof over the interval $[0, 2^n)$ can be done using inner product arguments over a committed value v , *i.e.*, $\text{PoK}\{(v, \rho) : C = g^v h^\rho \wedge (0 \leq v < 2^n)\}$. Two interval proofs can be combined as a range proof on the arbitrary range $[A, B]$ using a standard trick [15]. At a high level, for $2^{b-1} < B < 2^b$, the prover proves $v - A, v - B + 2^b$ are in $[0, 2^b)$. For SAC, we also set the upper limit (say, $[0, 2^{64} - 1]$) to cover possible scores.

Zero-Knowledge Argument of a Shuffle. A shuffle of commitments $\{C_1, \dots, C_N\}$ of messages $\{a_1, \dots, a_N\}$ is a set of commitments $\{C'_1, \dots, C'_N\}$ of $\{b_1, \dots, b_N\}$ committing to the same set of messages but in a permuted order, *i.e.*, $b_i = a_{\pi(i)}$ for some permutation $\pi : [N] \rightarrow [N]$. If we treat the messages as the roots of two polynomials of degree N , one can test for a random z (can be picked by the verifier) if $\prod_{i=1}^N (a_i - z) = \prod_{i=1}^N (b_i - z)$ holds for AoK of permutation. As an arithmetic circuit, this requires $2(N - 1)$ multiplication gates and is readily supported by the Bulletproofs. The prover and verifier computation are both linear in N but with logarithmic proof size (excluding the commitments). Appendix B reviews the ZK shuffle argument by Bayer and Groth [9] as an alternative.

Verifiable shuffle is used in SAC authentication after the client’s list of session identifiers is updated. Its purpose is to provide anonymity to the user.

BBS+ Signatures. BBS+ signature of Au, Susilo, Mu, and Chow [6] extends the BBS signature of Boneh, Boyen, and Shacham [12] with multiple message blocks and efficient ZK protocols for signing and verification. It is existentially unforgeable against adaptive chosen message attacks under the q -SDH assumption over pairing groups [16] with no efficient isomorphism between \mathbb{G}_1 and \mathbb{G}_2 .

Let $h_0, h_1, \dots, h_{\ell+1}$ be generators of \mathbb{G}_1 and f be a generator of \mathbb{G}_2 . The signer’s secret key is $\gamma \in \mathbb{Z}_p$ and the public key is $w = f^\gamma$. To sign on blocks of messages $(m_1, \dots, m_\ell) \in \mathbb{Z}_p^\ell$, the signer randomly picks $e, y \in \mathbb{Z}_p$ and computes $A = (h_0 h_1^{m_1} \dots h_\ell^{m_\ell} h_{\ell+1}^y)^{\frac{1}{\gamma+e}}$. The signature is (A, e, y) , and one can verify it by checking if $\hat{e}(A, w f^e) = \hat{e}(h_0 h_1^{m_1} \dots h_\ell^{m_\ell} h_{\ell+1}^y, f)$ holds.

Table 2. Major Notations for User-side and SP-side Data Structures

Notation	Description
x, q, s	user long-term secret key, nonce used for authentication, credential score
cred, σ	credential over $A = (x, q, s, \mathbb{U})$, signature from the SP, e.g., on A
\mathbb{U}	list of session identifiers (or tickets) kept by a user’s credential
\mathbb{L}	list of sessions ($(t_i, s_i, \sigma_i, \mathbb{I})$ tuples) maintained and published by the SP
$\mathbb{L}_I, \mathbb{U}_I$	sub-lists of \mathbb{L} or \mathbb{U} containing sessions of a specific status $I \in \{J, A, D\}$, meaning J udged, A ctive, or D ummy, respectively

BBS+ signatures mostly serve as credentials in many privacy-preserving systems. In SAC, they also link the ticket with its score.

Protocol \mathbb{P}_{Iss} . It allows a user to obtain a signature from the signer on a block of values (m_1, \dots, m_ℓ) without revealing them. It is used in SAC registration and credential update during authentication.

1. The user computes $C_M = h_1^{m_1} h_2^{m_2} \dots h_\ell^{m_\ell} h_{\ell+1}^{y'}$ for some randomly generated $y' \in \mathbb{Z}_p$, and sends C_M to the signer with the following proof:

$$\text{PoK} \left\{ (\{m_i\}_{i \in [1, \ell]}, y') : C_M = h_1^{m_1} h_2^{m_2} \dots h_\ell^{m_\ell} h_{\ell+1}^{y'} \right\}.$$

2. The signer aborts if the proof does not verify; otherwise, randomly picks $e, y'' \in \mathbb{Z}_p$, computes $A = (h_0 C_M h_{\ell+1}^{y''})^{\frac{1}{e+y'}}$, and returns (A, e, y'') to the user.
3. The user aborts if $\hat{e}(A, wf^e) \neq \hat{e}(h_0 h_1^{m_1} \dots h_\ell^{m_\ell} h_{\ell+1}^{y'+y''}, f)$. Otherwise, the user outputs σ as $(A, e, y = y' + y'')$.

Protocol \mathbb{P}_{Sig} . It enables proving the knowledge of a signature $\sigma = (A, e, y)$ on message blocks (m_1, \dots, m_ℓ) without revealing the signature nor the messages. The prover can also disclose messages contained in $\mathbb{D} \subset \{m_1, \dots, m_\ell\}$. The prover randomizes the signature with $\rho_1 \in \mathbb{Z}_p^*$ by setting $A' = A^{\rho_1}$. It also computes $b = h_0 h_{\ell+1}^s \prod_{i=1}^\ell h_i^{m_i} = A^{\gamma+e}$, $\bar{A} = A'^{-e} \cdot b^{\rho_1}$, and $\rho_3 = 1/\rho_1$. It then picks $r_2 \in \mathbb{Z}_p$ and sets $d = b^{\rho_1} \cdot h_{\ell+1}^{-\rho_2}$ and $s' = s - \rho_2 \cdot \rho_3$. The prover computes:

$$\text{PoK} \left\{ (\{m_j\}, e, \rho_1, \rho_2, \rho_3, s') : \bar{A}/d = h_{\ell+1}^{\rho_2} / A'^e \wedge h_0 \prod_{m_i \in \mathbb{D}} h_i^{m_i} \prod_{m_j \notin \mathbb{D}} h_i^{m_j} = d^{\rho_3} h_{\ell+1}^{-s'} \right\}.$$

The above proof is performed as follows:

1. The prover picks $r_e, r_1, r_2, r_3, r_{s'}, r_{m_1}, \dots, r_{m_\ell} \in \mathbb{Z}_p$ and sends to the verifier $R_1 = A'^{-r_e} h_{\ell+1}^{r_2}$ and $R_2 = d^{r_3} h_{\ell+1}^{-r_{s'}} \prod_{m_j \notin \mathbb{D}} h_i^{-r_{m_j}}$.
2. The verifier returns a random challenge $c \in_R \mathbb{Z}_p$.
3. The prover sends $z_e = r_e - ce$, $z_{s'} = r_{s'} - cs'$, $z_i = r_i - c\rho_i$ for $i \in [1, 3]$, and $z_{m_j} = r_{m_j} - cm_j$ for j such that $m_j \notin \mathbb{D}$.

4. The verifier checks $\frac{\bar{A}}{d} = R_1^c A^{-z_c} h_{\ell+1}^{z_2}, h_0 \prod_{m_i \in \mathbb{D}} h_i^{m_i} = R_2^c h_{\ell+1}^{-z_{s'}} / \prod_{m_j \notin \mathbb{D}} h_i^{z_{m_j}}$.

The proof consists of (A', \bar{A}, d, π) and can be verified by checking $A' \neq 1_{\mathbb{G}_1}$ and $\hat{e}(A', w) = \hat{e}(\bar{A}, f)$. The signer needs to publish $(\bar{g}, \bar{g}^\gamma)$ for $\bar{g} \neq 1_{\mathbb{G}_1}$ (for the zero-knowledge simulator). It is used in SAC authentication for proving the credential on attribute sets and proving the score of each authenticated session.

3.2 Key Ideas

Setup. In Setup, the SP runs the key generation of all underlying signature schemes (the choices will be specified in Sect. 3.3). All the signing keys are put to the secret key \mathbf{sk} , and the public keys are put to the public parameter \mathbf{pp} , along with the list \mathbb{L} , and system parameters including the threshold score s_{th} , score $-S_{\text{max}}$ for blocklisting, and the buffer sizes \mathbb{K} with K_{max} being the maximum.

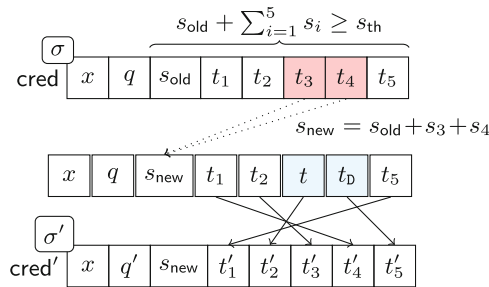


Fig. 1. Suppose t_3 is judged and t_4 is dummy. The user authentication now redeems $\mathbb{T} = \{t_3, t_4\}$ to get a new score s_{new} . A ticket t for this session is added to the credential. The user (with help from the SP) adds a dummy ticket t_D to maintain the credential size. The user proves that (t_1, t_2, t, t_D, t_5) is shuffled to $(t'_1, t'_2, t'_3, t'_4, t'_5)$ where the SP remains oblivious to the session type of all the t_i 's and information like whether they are old.

List \mathbb{L} contains tuples for the sessions, which includes:

- a set \mathbb{L}_D of dummy sessions, each with a score of 0,
- a set \mathbb{L}_A for storing active sessions with (dynamically) rated scores, and
- a set \mathbb{L}_J for judged sessions with finalized, judged scores.

Table 2 lists the major notations. Only \mathbb{L} in the table is public. Each entry of \mathbb{L} is of the form $(t_i, s_i, \sigma_{t_i, s_i}, \mathbf{state})$, storing a session identifier t_i , its score s_i , a signature σ_{t_i, s_i} (for the integrity of \mathbb{L}), and its state information \mathbf{state} , which is

- an empty string if the state is *active*, or
- a signature on t_i for *judged* or *dummy* (so later authentication can redeem it).

Suppose $s_{\text{th}} = 0$. Initially, the SP puts K_{max} dummy sessions of score 0, each in the form of $(t_i, 0, \sigma, \hat{\sigma})$, into \mathbb{L}_D , where σ signs on $(t_i, 0)$ and $\hat{\sigma}$ signs on t_i .

Registration. In **Reg**, each eligible user chooses a credential size $K \in \mathbb{K}$, randomly chooses two \mathbb{Z}_p values as the long-term secret x and the (first) nonce q for showing the freshness of the credential, and prepares an attribute set of size $(K + 3)$ as $A = (x, q, s = 0, t_1, \dots, t_K)$, where s is the (initial) score of the user, and $\mathbb{U}_D = (t_1, \dots, t_K)$ is a subset of dummy session identifiers from \mathbb{L}_D in **pp**.

Let \mathbb{U} denote the set of session identifiers kept by a credential. Just after registration, $\mathbb{U} = \mathbb{U}_D$. This will make a newborn credential indistinguishable (in size) from ones with the same number of sessions while some of them are active or judged. The user proves in **ZK** that attribute set A is well-formed:

- Knowledge of x, q , which are hidden from the SP;
- $s = 0$ (or any base score agreed between the user and the SP);
- (t_1, \dots, t_K) are dummy tickets, by showing $t_i \in \mathbb{L}_D$ (via PoK of signatures).

The SP then completes the signature issuance via the **ZK** signature issuance protocol \mathbb{P}_{Iss} , which produces a signature σ_A on A , serving as a user credential **cred**.

Authentication. In **Auth**, the user **ZK**-proves that credential **cred** on attribute set $A = (x, q, s, t_1, \dots, t_K)$ satisfies the access policy (to be made explicit in Sect. 3.3), except q is revealed in clear. Let $\mathbb{U} = (t_1, \dots, t_K)$. The proof consists of:

- *Credential Validation.* The user **ZK**-proves via \mathbb{P}_{Sig} that it has a signature σ_A on A , but reveals nonce q to show that it is not a replay of past credentials.
- *Score Satisfaction.* For set \mathbb{U} of size K in A certified by **cred**, the user proves in **ZK**, for each $t_a \in \mathbb{U}$, that the session t_a has score s_{t_a} as specified in list \mathbb{L} by proving via \mathbb{P}_{Sig} $\sigma_{t_a, s_{t_a}}$ is a signature on (t_a, s_{t_a}) . This proof can be done in a batch for efficiency. The user can then prove in **ZK** that $s + \sum_{t \in \mathbb{U}} s_t > s_{\text{th}}$ for some agreed threshold s_{th} . The SP cannot learn $\mathbb{U} = \{t_a\}$.

After some authentications and before any judgment is finalized, \mathbb{U} transforms to $\mathbb{U}_A \cup \mathbb{U}_D$, with some active tickets added. Over time, the SP will mark some sessions as judged. \mathbb{U} then transforms to $\mathbb{U}_J \cup \mathbb{U}_A \cup \mathbb{U}_D$ (similar to the SP-side public session list $\mathbb{L} = \mathbb{L}_J \cup \mathbb{L}_A \cup \mathbb{L}_D$). Also, a user may adjust \mathbb{U}_D to make the size of \mathbb{U} equal to some other allowed value $K' \in \mathbb{K}$.

The protocol should also ensure that the credential is updated faithfully:

- *Credential Update.*
 1. The SP chooses and publishes t' as the current session identifier.
 2. The user picks a set of tickets $\mathbb{T} \subseteq \mathbb{U}$ and uses the set membership proof \mathbb{P}_{Set} to **ZK**-prove that they are either judged or dummy, e.g., $\mathbb{T} \subset \mathbb{L}_J \cup \mathbb{L}_D$.
 3. The user updates \mathbb{U} to $\mathbb{U}' = (\mathbb{U} \setminus \mathbb{T}) \cup \mathbb{U}'_D \cup \{t'\}$, where $\mathbb{U}'_D \subset \mathbb{L}_D$ pads⁴ the number of sessions of \mathbb{U}' to K' , for a possibly new size K' (for $K' \in \mathbb{K}$).
 4. The user picks a fresh random nonce q' and proves in **ZK** that the shuffled \mathbb{U}' and the new aggregated score s' are well-formed w.r.t. sessions in \mathbb{T} .

⁴ Dummy sessions are judged with a default (zero) score, allowing users to pad \mathbb{U} with some dummy tickets from \mathbb{U} to \mathbb{T} to hide the judged ones among them.

5. The SP and user engage in credential issuance protocol \mathbb{P}_{Iss} for a new credential cred' on the new attributes $\mathbf{A}' = (x, q', s', \mathbf{U}')$.

For the new \mathbf{U}' with $\mathbb{U}'_{\mathbb{D}}$ and t' added and \mathbb{T} removed, verifiable shuffle is applied to permute their order, *i.e.*, the positions of \mathbb{T} in \mathbb{U} . Figure 1 depicts an example where the updated ticket set $(t_1, t_2, t, t_{\mathbb{D}}, t_5)$ is verifiably shuffled.

Finally, the SP updates pp by adding an entry $(t', 0, \sigma_{t',0}, \emptyset)$ into list \mathbb{L} , meaning that session t' has an initial score 0 (or any default) and an active state.

Session Score Update. In **Update**, the SP rates a session t by updating an entry $(t, s_{\text{old}}, \sigma_{t,s_{\text{old}}}, \cdot)$ in list \mathbb{L} into $(t, s_{\text{new}}, \sigma_{t,s_{\text{new}}}, \cdot)$, meaning that the new score of the session t is s_{new} , which can be less than s_{old} , negative, or even $-(S_{\text{max}} + 1)$ for revoking a credential, where S_{max} is the maximum possible score of any credential⁵ (and the threshold for maintaining unrevoked status is 0). To finalize a session t , the SP updates $(t, \cdot, \cdot, \emptyset)$ to $(t, \cdot, \cdot, \hat{\sigma})$ in \mathbb{L} , where $\hat{\sigma}$ is a signature on t .

A user can keep a ticket t in its credential as long as it can provide a valid membership proof of t against $\mathbb{L}_{\mathbf{A}}$. If t is finalized, the score for t in both $\mathbb{L}_{\mathbf{A}}$ and $\mathbb{L}_{\mathbf{J}}$ should be consistent. This way, the user credential score is correctly accounted for during authentication, and keeping tickets would not harm the system.

3.3 Instantiation

$\text{Setup}(1^\lambda) \rightarrow (\text{pp}, \text{sk})$: On input of the security parameter λ , the SP generates public parameter pp and the secret key sk as follows:

1. The SP chooses the bilinear map context over groups of prime order p (a $\text{poly}(\lambda)$ -bit prime) with pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.
2. The SP creates the key for BBS+ signature by randomly picking $\gamma \in \mathbb{Z}_p$, generators $h_0, h_1, \dots, h_{K_{\text{max}}+4} \in \mathbb{G}_1$ and $f_0 \in \mathbb{G}_2$, and computing $w_0 = f_0^\gamma$.
3. The SP creates the keys for set membership proofs by randomly picking $\beta \in \mathbb{Z}_p$, generators $g \in \mathbb{G}_1$, $f_1 \in \mathbb{G}_2$, and computing $w_1 = f_1^\beta$.
4. The SP maintains a list of sessions $\mathbb{L} = \{(t, s, \sigma, \text{state})\}$, where t is the session identifier, s is the score, σ is the BBS+ signature on (t, s) , and state is the status of the session (either an empty string for *active*, or a Boneh-Boyer signature on t , for *judged* or *dummy*). The SP initializes \mathbb{L} by adding K_{max} dummy sessions $(t, 0, \sigma, \hat{\sigma})$ with a different identifier t , where:
 - $\sigma = (A = (h_0 h_1^t h_3^t)^{\frac{1}{\gamma+e}}, e, y)$ is a BBS+ signature for random $e, y \in \mathbb{Z}_p$.
 - $\hat{\sigma} = g^{\frac{1}{\beta+e}}$ denotes a Boneh-Boyer signature on t .
5. The SP runs $\mathcal{K}(1^\lambda)$, the algorithm for generating the common reference string crs for the ZKPoK protocol PoK (to be defined below). In particular, it contains the set of Boneh-Boyer signatures $\{\hat{\sigma}_i\}$ needed by the signature-based membership proof \mathbb{P}_{Set} .
6. The SP sets secret key $\text{sk} = (\gamma, \beta)$ and public parameters

$$\text{pp} = (\text{crs}, g, h_0, \dots, h_{K_{\text{max}}+4}, f_0, f_1, w_0, w_1, \mathbb{K}, S_{\text{max}}, \mathbb{L}).$$

⁵ The SP can derive it from K_{max} with an appropriate upper limit for each session.

$\text{Reg}(\mathcal{U}(\text{pp}), \mathcal{S}(\text{pp}, \text{sk})) \rightarrow \{\mathcal{U}(\text{cred}, \text{A}), \mathcal{S}(\text{pp}')\}$: The user obtains a credential from the SP via an authenticated channel as follows:

1. (Preparation:) The user randomly picks $x', q \in \mathbb{Z}_p$, and selects $K \in \mathbb{K}$. The user prepares attributes of size $K + 3$ as $\text{A} = (x', q, s = 0, t_1, \dots, t_K)$, where $t_i \in \mathbb{L}_D$ for $i \in [1, K]$. The user generates the commitment $C'_M = h_1^{x'} h_2^q h_4^{t_1} \dots h_{K+3}^{t_K} h_{K+4}^{y'}$ for some random $y' \in \mathbb{Z}_p$ and sends C'_M to the SP.
2. (Signing:) The SP randomly picks $x'' \in \mathbb{Z}_p$, computes $C_M = C'_M \cdot h_1^{x''}$, and sends x'' to the user. The user secret is then computed by $x = x' + x''$. They then engage in the protocol \mathbb{P}_{iss} of the BBS+ signature for the commitment C_M using the public key $(h_0, \dots, h_{K+4}, f_0, w_0)$. In the end, the user obtains a BBS+ signature σ_A . The public parameter $\text{pp}' = \text{pp}$ remains unchanged.
3. (Credential Generation:) The user stores the credential $\text{cred} = \sigma_A$ and the attributes $\text{A} = (x, q, s = 0, \mathbb{U} = \{t_1, \dots, t_K\})$.

$\text{Auth}(\mathcal{U}(\text{pp}, \text{cred}, \text{A}, f), \mathcal{S}(\text{pp}, \text{sk}, f)) \rightarrow \{\mathcal{U}(b, \text{cred}', \text{A}'), \mathcal{S}(b, \text{pp}')\}$: With cred on attributes $\text{A} = (x, q, s, \mathbb{U} := \{t_1, \dots, t_K\})$, the user attempts to prove that a credential with a score above s_{th} given in the access policy f as follows:

1. (Nonce revelation:) The user reveals q privately to the SP. If q is fresh, the SP picks t' and publishes it as the identifier of the *new (active) session* for this user authentication. Otherwise, the SP aborts as it is a replay.
2. (Proof about authentication requirements:) To prove to the SP, the user runs a combined ZKPoK PoK, which is a conjunction of several ZKPoK's:
 - (a) \mathbb{P}_{Sig} : knowing a signature σ_A on the attributes $\text{A} = (x, q, s, \{t_1, \dots, t_K\})$;
 - (b) \mathbb{P}_{Sig} : knowing $s_{t_i}, \sigma_{t_i, s_{t_i}}$ where $\sigma_{t_i, s_{t_i}}$ is a BBS+ signature on the ticket t_i and the score s_{t_i} , for $t_i \in \mathbb{U}$;
 - (c) Range proof⁶: $s_{\text{th}} \leq s + \sum_{i=1}^K s_{t_i} \leq S_{\text{max}}$.
3. (Updated credential:) The combined ZKPoK PoK also proves about knowing:
 - (a) (many copies of) \mathbb{P}_{Set} : a set of *judged* or *dummy* session identifiers \mathbb{T} , via a signature-based membership proof that for all $t_i \in \mathbb{T}$, there exists a Boneh-Boyer signature $\hat{\sigma}_i$ (only presents for a judged or dummy session);
 - (b) commitment on a new nonce $q' \in \mathbb{Z}_p$;
 - (c) $\hat{C}_i = g^{t_i} h_0^{s_{t_i}} \forall t_i \in \mathbb{U}'$, where $(\mathbb{U}', s') \leftarrow \text{Redeem}(\mathbb{U} \cup \{t'\}, \mathbb{T}, K')$ to be defined shortly, and $K' \in \mathbb{K}$, which the user can choose to have $K' = K$.
 - (d) \hat{C}'_i is a shuffle π of \hat{C}_i , which means $\hat{C}'_i = g^{t_{\pi(i)}} h_0^{s'_{t_i}}$ for all $t_i \in \mathbb{U}'$;
 - (e) C'_M is a new commitment on $(x, q', s + s', \hat{\mathbb{U}})$, where $\hat{\mathbb{U}} = \{t_{\pi(1)}, \dots, t_{\pi(K')}\}$.

$\text{Redeem}(\mathbb{U}, \mathbb{T}, K)$ is for redeeming scores in \mathbb{T} within \mathbb{U} and puts dummy sessions into \mathbb{U}' if needed to make $|\mathbb{U}'| = K'$. Note that $\mathbb{U}' = \mathbb{U} \cup \{t'\} \setminus \mathbb{T}$ is of size $|\mathbb{U}'| = K + 1 - |\mathbb{T}|$. If it is less than K' , \mathbb{U}' will be padded with dummy sessions $\{d\} \subseteq \mathbb{L}_D$. It also outputs a new score $s' = \sum_{t_j \in \mathbb{T}} s_{t_j}$.

⁶ It involves commitments of s and s_{t_i} for $t_i \in \mathbb{U}$ if it is instantiated by Bulletproofs.

4. (New credential generation:) The SP issues a BBS+ signature cred' on $A' = (x, q', s + s', \hat{U})$ by using the protocol \mathbb{P}_{Iss} on C'_M .
5. (List update:) The SP adds the entry $(t', 0, \sigma, \emptyset)$ to list \mathbb{T} , where σ is a BBS+ signature on the message $(t', 0)$.

Appendix C provides a concrete instantiation using signature-based range proof \mathbb{P}_{Set} , verifiable shuffle [9], and batch BBS+ signatures (in Appendix A).

$\text{Update}(\text{pp}, \text{sk}, \text{aux}) \rightarrow \text{pp}'$: To assign a new score s to a session t given in aux , the SP issues a new BBS+ signature σ on (t, s) for the new or updated entry (t, s) in \mathbb{L} . (See Sect. 3.5 for the signature timestamping issue.) To block, the SP assigns the lowest possible negative score $-(S'_{\max} + 1)$.

When no more change is needed for a session t , the SP changes the status of the session to “judged” by computing $\hat{\sigma}_t = g^{\frac{1}{\beta+1}}$ and putting $(t, \cdot, \cdot, \hat{\sigma}_t)$ on \mathbb{L} .

3.4 Efficiency and Flexibility Highlights

Achieving Efficiency for Many Active Sessions. The user needs to store all its active session identifiers and construct proof about them for claiming the scores of all the sessions, *i.e.*, the computation complexity grows with the size of user-side storage. We use a few interesting techniques to improve user-side efficiency:

1. Users can remove session identifiers from the credential once they are judged. Their scores will be permanently aggregated to the finalized score field s of the credential. Low-usage users can choose to keep a small buffer size K .
2. The score of each session is signed by BBS+ signatures, which support efficient ZK proof (without proving pairing relation) and batch verification.
3. The same proof asserts the ticket status without disjunctive proof and the involved commitments needed by PERM/PEREA for proving unjudged sessions ($t_i - \text{jp} < N$ or knowledge of a judgment on t_i). This saves approximately K range proofs without requiring sequential judging.

Fair Rate-Limiting. SAC supports a (set of) maximum buffer size K (\mathbb{K}). The SP is required to finalize the judgment on at least one of these sessions when all K active session slots are used up by the user. Any user is only rate-limited by its own previously unjudged tickets. Besides, the number of unjudged tickets sets an upper bound to the cardinality of the to-redeem ticket set \mathbb{T} . PERM/PEREA can be treated as a special case where $|\mathbb{T}| = 1$ (“redeeming” the first ticket).

Removal of Old Judgments. Signatures that were too old could be removed. The SP can still keep only the session identifier and score. If an infrequent user eventually claims those sessions, it can still be done without affecting correctness but just anonymity. This is more flexible than the existing blacklist-based approach, in which truncating the blacklist may forgive some bad users for free.

3.5 Discussion on Privacy and Security Issues

Variations of User Authentication/Redemption Behavior. The SP and the users can judge and redeem scores from the sessions, respectively, in an arbitrary order. From an anonymity perspective, suppose a user always redeems the first few sessions in its credential during authentication while another user always redeems the last few; their difference in behavior may compromise anonymity. Verifiable shuffle is thus used to ensure obliviousness to any pattern of redemption. Anonymity holds among users using the same size parameters, specifically, K (the number of sessions in the credential) and $|\mathbb{T}|$ (the size of the ticket to redeem) with the system-wide choices of size \mathbb{K} and many dummy tickets \mathbb{L}_D .

Retrieval of Signatures. Similar to redemption in PERM, users should also retrieve the signatures corresponding to others' sessions to hide in the crowd. The download can be amortized, such as getting a random subset after each interaction (e.g., browsing on a forum), or can be done via private information retrieval, like what is necessary for other privacy-preserving applications (e.g., getting the list of bridges for Tor [40]). Compared with BLAC/EPID, which performs expensive cryptographic operations against each blocklist entry, our "allowlist" does not need to be downloaded during the authentication.

Credential "Hijacking" Prevention. A user can freely choose the nonce for each credential update. If a user picks a used nonce, it cannot be used to authenticate. This brings a subtle issue if a malicious user somehow learns the victim's credential nonce. A malicious user can block a victim user by using the victim's nonce as the nonce of a new credential and authenticate using it before the victim.

The issue can be prevented by slightly extending the protocol so the SP can contribute a part of the nonce randomness. The SP and user respectively pick q' , q'' . The new nonce will be set to $q' + q''$ during credential generation. The probability of hitting a used nonce is negligible if the nonce space is exponentially large, which is our case. In the rare event that results in a used nonce, the user can reveal q'' and rerun the credential generation part with the SP.

Signatures Timestamping/Expiry. SAC needs to expire old signatures when the score is updated. This is a general problem of timestamping signatures, which has been studied thoroughly in different contexts with multiple solutions, e.g., outsourced and authenticated data structure which supports membership query and update [42]. Solutions for timestamping signatures can be plugged into SAC generically. We describe below a simple (but not necessarily optimal) solution.

The SP issues a new set of signatures for each session in predefined intervals. The interval identifier can be signed together with the updated score via the multi-block feature of BBS+ signatures. The proof will require proving the interval to be the current one, as how proofs are made on other messages certified.

This simple solution requires the SP to generate multiple signatures for multiple updates. Nevertheless, the SP is supposed to be more resourceful and can

use online/offline signatures with preparation done offline. The SP would not update indefinitely as the scores of active sessions will eventually be finalized. Only the most-updated signatures or the finalized ones should be kept.

Expiry of signatures is only needed for volatile scores. If a session can only be changed from unjudged to finalized, signature expiry is not needed at all.

3.6 Integration with Other System Components

Precondition for Registration. To prevent unauthorized sharing of credentials, the standard practice is to embed valuable secrets to credentials such that sharing them means sharing the secret too. The SP can require the user to post the public key of a cryptocurrency account, prove the knowledge of its secret key to the SP, and use it as the user secret in the SAC credential.

Multi-SP and Decentralization. A 1-out-of- n disjunctive (or membership) proof can be used to prove the validity of cred and judgments under a set of authorized SPs' public keys. With homomorphic commitments, judgments from different SPs can be combined arithmetically. However, one cheating SP can ruin the system by issuing malformed credentials. Threshold BBS+ signature [30] with blind signing protocol can be applied to achieve t -out-of- n threshold signing, where t -out-of- n signing parties (SPs) are needed to issue/update user credentials.

Enforcement vs. Voting. SAC is for privacy-preserving reputation *enforcement* (PPRE). The basic setting assumes the SP decides the score. Alternatively, privacy-preserving reputation *voting* (PPRV), often called “privacy-preserving reputation” [33], aims for the privacy of (peer) voters who cast votes of a score for other users. They may use cryptographic techniques such as linkable ring signatures [25, 32] for double-voting detection [24]. Some require a TTP, or users can request votes from many voters via secure multi-party computation.

Among many PPRV approaches, “reputation transfer” [33] overlaps with PPRE. A user can use a claiming mechanism to link scores cast by voters to a new pseudonym. Nevertheless, users are not forced to claim every (negative) rating. Some work discusses the usage of ZKP, which should cover all ratings and brings us back to linearly processing a global list we strive to avoid. In general, incorporating PPRV with PPRE may not be straightforward.

A concurrent work [23] outlined a “decentralized anonymous social networks” construction supporting both PPRE and PPRV, with a focus on sustainability.

Keyed-Verification Anonymous Credentials (KVAC). In many scenarios, the SP also acts as the verifier without the need for public verifiability offered by SAC. In KVAC [21], verifying the proof of possession of a credential requires the issuer's secret (issuance) key. KVAC [8, 21, 27] does not consider credential updates based on previously committed (and authenticated) messages. Moreover, revocable KVAC [34] assumes a TTP revocation authority using traceable-tag techniques [1, 6, 22]. Finally, KVAC [8, 21, 27] could still use public-key algebraic-group operations. We leave more efficient constructions using KVAC as future work.

4 Performance

4.1 Computation and Communication Complexities

We select PERM with one score category ($\ell = 1$) [4] as the representative to compare. We set $|\mathbb{T}| = 10$ (the set of tickets to be redeemed). New dummy sessions will be appended and shuffled with currently active sessions to maintain the credential buffer size. The size for the shuffle proof via Bulletproofs consists of the input and output commitments and the logarithmic size (inner-product) proof (1601 bytes for $K = 200$). An authentication needs to prove $(K + 3)$ BBS+ signatures for the accumulated score. PERM needs extra K (possibly simulated) range proofs for its ZKP of partial knowledge. Since we utilize succinct (sublinear size) ZKAoK, the communication overhead is relatively small compared to the ZKP on the BBS+ signatures (a few KBs versus hundreds of KBs).

4.2 Empirical Results

We run SAC and PERM [4] over a desktop PC equipped with Ryzen 7 3700X and 96 GB RAM running Ubuntu 20.04 (on Windows Subsystem for Linux 2). We modified the Rust libraries⁷ that implemented the BBS+ signature over the pairing-friendly curve BLS12-381, and the Bulletproofs implementation⁸ for BLS12-381. For SAC, the set of possible buffer sizes is $\mathbb{K} = \{10, 30, \dots, 200\}$. We compare SAC with running PERM on different window sizes $K_{\max} = K$.

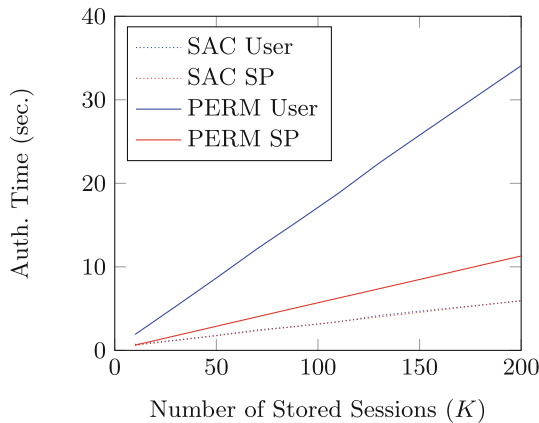


Fig. 2. Computation time of the service provider and user with $|\mathbb{T}| = 10$: (i) PERM only supports redeeming the first ticket, *i.e.*, $|\mathbb{T}| = 1$. (ii) BBS+ signatures over Type-3 curves feature batch ZKP verification, which our implementation does not employ.

⁷ <https://github.com/docknetwork/crypto>.

⁸ <https://github.com/dalek-cryptography/bulletproofs>.

Figure 2 outlines the authentication time. For the user side, SAC takes 0.714s/5.91s when $K = 10, 200$. Meanwhile, PERM takes 1.90s and 34.0s when $K = 10, 200$. For $K = 50$, SAC still performs better than PERM at $K = 10$.

For the SP side, SAC takes 0.612s/5.97s when $K = 10/200$. Comparatively, PERM takes 0.625s and 11.3s when $K = 10, 200$. For $K \geq 90$, the SP computation cost for PERM is like twice of SAC for the same number of stored sessions.

For data transfer, we suppose each session score and the maximum reputation score of a user are in a 64-bit range. Furthermore, the total number of authentications is less than 2^{64} . The serialized size of a \mathbb{Z}_p , \mathbb{G}_1 , and \mathbb{G}_2 element is 32, 48, and 96 bytes, respectively. Each entry in our list \mathbb{L} is 176 bytes for active sessions and 80 bytes for judged or dummy sessions. The total downlink complexity is $176|\mathbb{L}_A| + 80(|\mathbb{L}_D| + |\mathbb{L}_J|)$ bytes. Here, the signature of the SP used in the range proof is put in the public parameters as in PERM. Dummy sessions in SAC never change. Users can download them during registration.

To compare with PERM, suppose the number of anonymous authentications per day is 20,000, and the same number of authentications is judged (which only adds an extra 80 bytes to \mathbb{L}). A user of PERM and SAC would need to download respectively 3.5MB and 5MB of data a day to keep up-to-date without decoys.

For each authentication, a user of both PERM and SAC proves possession of $O(K)$ BBS+ signature on stored sessions with the scores accumulated, which is about 368 KBytes for $K = 200$. PERM runs ZKP for K disjunctive statements, which consist of signature possession and range proof. We instantiated the range proof with Bulletproofs; the extra communication overhead of PERM is around 3676 bytes per session. On the other hand, the proof of shuffle using Bulletproofs in SAC is 1601 bytes (for $K = 200$) and the total size with commitments is $96 \cdot K$ bytes. The extra ZKP on a BB signature has around $240 \cdot |\mathbb{T}|$ bytes ($|\mathbb{T}| \leq K$). Thus, we have a lower authentication communication cost than PERM.

5 Conclusion

We propose scored anonymous credentials (SAC), a new and intuitive credential design supporting revocation and reputation. Unlike the two existing designs of either checking every session that ever happened (e.g., BLAC(R) [5, 44]) or assuming sequential judgments (e.g., PEREA [45], PERM), we directly deal with the user sessions via verifiable shuffle and other optimized cryptographic techniques. We also support downgrading the score of a session until it is finalized, while existing updatable anonymous credentials (e.g., PE(AR)² [47], PERM) only support score upgrade. We evaluate the efficiency of our proposed system SAC and show that it outperforms existing related systems, given the simplicity of our design and cryptographic techniques. We thus resolved the open problem of devising an anonymous credential system with (reputation and) revocation mechanism that does not halt the entire system due to just one misbehaving user.

A Batch BBS+ Signature

The original ZKP for a BBS+ signature [6] is performed as follows.

Protocol \mathbb{P}_{Sig} . It allows a prover to prove that it knows a signature $\sigma = (A, e, y)$ on blocks of messages (x_1, \dots, x_ℓ) without revealing the signature nor the messages.

1. The prover randomly generates $r_A \in \mathbb{Z}_p$, sets $\beta = r_A e$, and sends $A_1 = A \hat{h}^{r_A}$, $A_2 = h_1^{r_A}$ to the verifier along with the following proof Π :

$$\text{PoK} \left\{ \begin{array}{l} (\{x_i\}_{i \in [1, \ell]}, e, y, r_A, \beta) : \\ (A_2 = h_1^{r_A}) \wedge (1 = A_2^{-e} h_1^\beta) \wedge \\ \frac{\hat{e}(A_1, w)}{\hat{e}(h_0, f)} = \hat{e}(h_1, f)^{x_1} \cdots \hat{e}(h_\ell, f)^{x_\ell} \cdot \hat{e}(h_{\ell+1}, f)^y \cdot \\ \hat{e}(\hat{h}, w)^{r_A} \cdot \hat{e}(\hat{h}, f)^\beta / \hat{e}(A_1, f)^e \end{array} \right\}.$$

2. Upon receiving (A_1, A_2, Π) , the verifier outputs 1 if proof Π is valid.

Details such as its construction can be found in [6].

Batch BBS+ Signatures. We construct batch BBS+ (B-BBS+) featuring a batch verification algorithm for BBS+ signatures and its zero-knowledge version. Without loss of generality, we illustrate with two-message blocks $(s_1, m_1) \in \mathbb{Z}_p^2$ (suffices for SAC). The signature $(A_1 = (h_0 h_1^{s_1} h_2^{m_1} h_3^{y_1})^{\frac{1}{7+e_1}}, e_1, y_1)$ can be verified by:

$$\hat{e}(A_1, w f^{e_1}) = \hat{e}(h_0 h_1^{s_1} h_2^{m_1} h_3^{y_1}, f).$$

For $i \in [1, K]$, let (A_i, e_i, y_i) be the signature on (s_i, m_i) . To batch verify, pick $(\delta_1, \dots, \delta_K)$ as a random vector of ℓ_b -bit elements from \mathbb{Z}_p and test if:

$$\hat{e}\left(\prod_{i=1}^K A_i^{\delta_i}, w\right) \cdot \hat{e}\left(\prod_{i=1}^K A_i^{e_i \delta_i}, f\right) = \hat{e}\left(h_0^{\sum_{i=1}^K \delta_i} \cdot h_1^{\sum_{i=1}^K s_i \delta_i} \cdot h_2^{\sum_{i=1}^K m_i \delta_i} \cdot h_3^{\sum_{i=1}^K y_i \delta_i}, f\right).$$

which takes 2 pairings, $2K + 4$ exponentiations in \mathbb{G}_1 , and $2K + 2$ multiplications in \mathbb{G}_1 . Since 1 pairing takes roughly the time of 5 exponentiations, and non-batch verifications take $2K$ pairings, $4K$ exponentiations in \mathbb{G}_1 , and $4K$ multiplications in \mathbb{G}_1 for K signatures, the batch verification speeds up by $4.5 \times$ for large K .

Theorem 1. *For security level ℓ_b , the above algorithm is a batch verifier for BBS+ signatures with the probability of accepting an invalid signature being $2^{-\ell_b}$.*

The batching technique for BBS+ basically follows from [31][Theorem 3.2]. BBS+ signature is weakly secure in the standard model under the q -SDH assumption [6]. Below is the batched zero-knowledge proof \mathbb{P}_{Bvfy} for B-BBS+.

Protocol \mathbb{P}_{Bvfy} . It allows proving the knowledge of K signatures $\sigma_i = (A_i, e_i, y_i)$ on messages (s_i, m_i) for $i \in [1, K]$.

- Let δ_i be an ℓ_i -bit element picked by the verifier in \mathbb{Z}_p . The prover randomly generates $\mu_i \in \mathbb{Z}_p$, computes $B_i = h_1^{\mu_i}$, $D_i = A_i \hat{h}^{\mu_i}$, $\iota_i = \mu_i e_i$, and sends B_i, D_i to the verifier along with the proof Π :

$$\text{PoK} \left\{ \begin{array}{l} (\{s_i, m_i, \mu_i, e_i, y_i, \iota_i\}_{i \in [1, K]}): \\ \bigwedge_{i=1}^K (B_i = h_1^{\mu_i} \wedge 1 = B_i^{-e_i} h_1^{\iota_i}) \wedge \\ \frac{\hat{e}(\prod_{i=1}^K D_i^{\delta_i}, w)}{\hat{e}(h_0^{\sum_{i=1}^K \delta_i}, f)} = \hat{e}(h_1, f)^{\sum_{i=1}^K \delta_i s_i} \cdot \hat{e}(h_2, f)^{\sum_{i=1}^K \delta_i m_i} \cdot \\ \hat{e}(h_3, f)^{\sum_{i=1}^K \delta_i y_i} \cdot \hat{e}(\hat{h}, w)^{\sum_{i=1}^K \delta_i \mu_i} \cdot \\ \hat{e}(\hat{h}, f)^{\sum_{i=1}^K \delta_i \iota_i} / \hat{e}(\prod_{i=1}^K D_i^{\delta_i e_i}, f) \end{array} \right\}.$$

- Upon $((B_1, D_1), \dots, (B_K, D_K), \Pi)$, the verifier outputs 1 if proof Π is valid.

B Alternative for Zero-Knowledge Argument of a Shuffle

Let $C_i = \mathcal{E}_{\text{pk}}(M_i; \rho_i)$ for $i \in [1, N]$, a ciphertext created from encrypting message M_i under public key pk using randomness ρ_i . Homomorphic encryption allows simple multiplications of encrypted M_i and M_j without decrypting it first via simple multiplication of ciphertexts: $\mathcal{E}_{\text{pk}}(M_i; \rho_i) \mathcal{E}_{\text{pk}}(M_j; \rho_j) = \mathcal{E}_{\text{pk}}(M_i M_j; \rho_i + \rho_j)$. There are efficient zero-knowledge arguments for showing a shuffling of ciphertexts produced by homomorphic encryption, *i.e.*, for all $i \in [N]$, decrypting C'_i is the same as decrypting $C_{\pi(i)}$, where $\pi : [N] \rightarrow [N]$ is the permutation.

We review a recent scheme by Bayer and Groth [9]. The first step is to commit to permutation π . The prover receives a challenge x and commits to $x^{\pi(1)}, \dots, x^{\pi(N)}$. The prover will give an argument of knowledge of openings of the commitments to permutations of $1, \dots, N$ and x^1, \dots, x^N . The prover demonstrates that the same permutation has been used in both cases using random challenges y and z . By using the homomorphic properties of the commitment, the prover can compute commitments to $d_1 - z = y\pi(1) + x^{\pi(1)} - z, \dots, d_N - z = y\pi(N) + x^{\pi(N)} - z$, in a verifiable manner, then uses a product argument to show:

$$\prod_{i=1}^N (d_i - z) = \prod_{i=1}^N (y_i + x^i - z).$$

These are two identical degree- N polynomials in z (with the roots permuted). By Schwartz-Zippel lemma, the prover has a negligible chance over the choice of z of making a convincing argument unless there is a permutation π such that $d_i = y\pi(i) + x^{\pi(i)}$ for $i \in [1, N]$. Furthermore, there is negligible probability over the choice of y of this being true unless the first commitment contains $\pi(i)$ and the second commitment contains $x^{\pi(i)}$ for $i \in [1, N]$.

The prover has commitments to $x^{\pi(i)}$ and uses the multi-exponentiation argument to show there exists a ρ such that

$$\prod_{i=1}^N C_i^{x^i} = \mathcal{E}_{\text{pk}}(1; \rho) \prod_{i=1}^N C_i^{x^{\pi(i)}}.$$

Since the encryption \mathcal{E} is homomorphic, the verifier can deduce that $\prod_{i=1}^N M_i^{x^i} = \prod_{i=1}^N M_i^{x^{\pi(i)}}$ for some permutation π . Since x is a random challenge chosen by the verifier, we have a correct shuffle with overwhelming probability.

Depending on the implementation, there is a trade-off between the round complexity, communication complexity, and the computation time of the users and the SP [9]. In principle, we can apply any zero-knowledge arguments of a shuffle. For efficiency, we will require the shuffle and the commitments (to session identifiers) to be in the same group. For our construction, we can use the homomorphic ElGamal encryption of a message M with $\text{pk} = (h, u)$ is (Mu^ρ, h^ρ) , which fits with Bayer-Groth’s argument [9]. Since decryption is not needed in our case, we can encode a ticket t by $M = h^t$, and only include the first part of the ciphertext. It is easy to see that it preserves the homomorphic property.

C Authentication using Batch-BBS+ and Range Proof

The signature-based range proof can be constant-size if the threshold score s_{th} and the maximum score s_{max} is short, e.g., ℓ_b -bit integers for $\ell_b = 10$. Let $g \in \mathbb{G}_1$, $u \in \mathbb{G}_1$ (for the shuffle), and $f_2 \in \mathbb{G}_2$ be generators. The SP picks a random $\alpha \in \mathbb{Z}_p$, and computes $w_2 = f_2^\alpha$. The SP puts BB signatures $Y_j = g^{\frac{1}{\alpha+j}}$ for all $j \in [s_{\text{th}}, s_{\text{max}}]$ in the public parameters. (One can instead apply Bulletproofs [14].)

1. Let $\sigma_A = (A, e, y)$ be the credential on attributes $A = (x, q, s, \{t_1, \dots, t_K\})$. Let \mathbb{J} be a set of $M = |\mathbb{J}|$ indexes where $t_j \in \mathbb{T}$ for all $j \in \mathbb{J}$. The user sends the parameters K, M , and q to the SP. The SP returns a new identifier t' .
2. The SP randomly picks $\delta_1, \dots, \delta_K, \zeta_i, \iota_i$ for $i \in [1, K - M]$ and ℓ_b -bits numbers $\{\theta_j\}_{j \in \mathbb{J}}$ and sends to the user.
3. The user picks $r_A, r_t, r_e, r_\beta, r_x, r_s, r_1, \dots, r_K$ in \mathbb{Z}_p , computes $\beta = r_A e$ and:

$$\begin{aligned} A_1 &= A \hat{h}^{r_A}, \quad A_2 = h_1^{r_A}, \quad T_1 = h_1^{r_t}, \quad T_2 = A_2^{-r_e} h_1^{r_\beta}, \\ R_x &= g^{r_x}, \quad R_s = g^{r_s}, \quad R_1 = g^{r_1}, \dots, R_K = g^{r_K}, \\ R &= \hat{e}(h_1, f)^{r_x} \cdot \hat{e}(h_3, f)^{r_s} \cdot \hat{e}(h_4, f)^{r_1} \dots \hat{e}(h_\ell, f)^{r_K} \\ &\quad \hat{e}(h_{\ell+1}, f)^{r_y} \cdot \hat{e}(\hat{h}, w)^{r_t} \cdot \hat{e}(\hat{h}, f)^{r_\beta} / \hat{e}(A_1, f)^{r_e}. \end{aligned}$$

For $i \in [1, K]$, let (A_i, e_i, y_i) be the B-BBS+ signature for ticket t_i and score s_i in \mathbb{L} . The user randomly picks $\mu_i, r_{s_i}, r_{\mu_i}, r_{\beta_i}$ in \mathbb{Z}_p , and computes

$$\begin{aligned} \beta_i &= \mu_i e_i, \quad D_i = A_i \hat{h}^{\mu_i}, \quad B_i = h_1^{\mu_i}, \quad S_i = g^{r_{s_i}}, \quad \bar{T}_i = h_1^{r_{\mu_i}}, \quad \bar{W}_i = B_i^{-r_{e_i}} h_1^{r_{\beta_i}}, \\ R_{\mathbb{L}} &= \hat{e}(h_1, f)^{\sum_{i=1}^K \delta_i r_{\mu_i}} \cdot \hat{e}(h_2, f)^{\sum_{i=1}^K \delta_i r_{s_i}} \hat{e}(h_3, f)^{\sum_{i=1}^K \delta_i r_{y_i}} \cdot \hat{e}(\hat{h}, w_0)^{\sum_{i=1}^K \delta_i r_{\mu_i}} \\ &\quad \hat{e}(\hat{h}, f)^{\sum_{i=1}^K \delta_i r_{\beta_i}} / \hat{e}\left(\prod_{i=1}^K D_i^{\delta_i r_{e_i}}, f\right). \end{aligned}$$

Let $s^* = s + \sum_{i=1}^K s_i$. The user computes a range proof of s^* with $v, r_v \in \mathbb{Z}_p$:

$$V = Y_s^v, \quad R^* = \hat{e}(V, f_2)^{-r_s - \sum_{i=1}^K r_{s_i}} \cdot \hat{e}(g, f_2)^{r_v}.$$

For all $j \in \mathbb{J}$, the user randomly picks v_j, r_{v_j} in \mathbb{Z}_p and computes:

$$V_j = \hat{\sigma}_j^{v_j}, \quad R_{\mathbb{J}} = \hat{e}\left(\prod_{j \in \mathbb{J}} V_j^{-\theta_j r_j}, f\right) \cdot \hat{e}(g, f)^{\sum_{j \in \mathbb{J}} \theta_j r_{v_j}}.$$

The user runs $\text{Redeem}(\mathbb{U} \cup \{t'\}, \mathbb{T}, K')$ as follows. Suppose w.l.o.g. $\mathbb{U} = \{t_1, \dots, t_K\}$ and $\mathbb{T} = \{t_{K-M+1}, \dots, t_K\}$. So $\mathbb{J} = \{K-M+1, \dots, K\}$. Suppose $\mathbb{U}' = \{t'_1, \dots, t'_{K'-1}, t'\}$ is selected where $t'_i = t_i$ for $i \in [1, K-M]$, $t'_{K-M+1} = t'$, and t'_i is a random element in \mathbb{L}_p , $i \in [K-M+2, K']$. The user computes $s' = \sum_{j \in \mathbb{J}} s_j$, and the encryption for tickets t'_1, \dots, t'_{K-M+1} , by randomly picking $\rho_i \in \mathbb{Z}_p$ for $i \in [1, K-M+1]$ and setting $\hat{C}'_{i,1} = g^{t'_i u^{\rho_i}}$, $\hat{C}'_2 = g^{\sum_{i=1}^{K-M+1} \zeta_i \rho_i}$. The user picks a permutation $\pi : [K'] \rightarrow [K']$. Let $\hat{t}_i = t'_{\pi(i)}$ for $i \in [1, K-M+1]$. The user also computes the homomorphic encryption by randomly picking $\rho'_i \in \mathbb{Z}_p$ and setting $\hat{C}'_{i,1} = g^{\hat{t}_i u^{\rho'_i}}$, $\hat{C}'_2 = g^{\sum_{i=1}^{K-M+1} \iota_i \rho'_i}$. For updating the credential, the user randomly picks $\hat{r}_1, \dots, \hat{r}_{K-M}$ and sets:

$$\begin{aligned} C'_M &= h_1^x h_2^{q'} h_3^{s+s'} h_4^{\hat{t}_1} \dots h_{K-M+4}^{\hat{t}_{K-M+1}} h_{K'+4}^{y'}, \\ R'_M &= h_1^{r_x} h_2^{r_{q'}} h_3^{r_s + \sum_{j \in \mathbb{J}} r_{s_j}} h_4^{\hat{r}_1} \dots h_{K-M+4}^{\hat{r}_{K-M+1}} h_{K'+4}^{r_{y'}}, \\ \hat{R}_1 &= g^{\hat{r}_1}, \quad \dots, \quad \hat{R}_{K-M+1} = g^{\hat{r}_{K-M+1}} \end{aligned}$$

and sends SP set \mathbb{J} , dummies $(t'_{K-M+2}, \dots, t'_{K'-1})$, and the commitments: $A_1, A_2, T_1, T_2, R_x, R_s, R_1, \dots, R_K, R, \{D_i, B_i, S_i, \bar{T}_i, \bar{W}_i\}_{i \in [1, K]}, R_{\mathbb{L}}, V, R^*, \{V_j\}_{j \in \mathbb{J}}, R_{\mathbb{J}}, \{\hat{C}'_{i,1}, \hat{C}'_{i,1}, \hat{R}_i\}_{i \in [1, K-M+1]}, \hat{C}'_2, \hat{C}'_2, C'_M, R'_M$.

The user can now perform the ZK argument for shuffling $\hat{C}'_{i,1}, \hat{C}'_2$ to $\hat{C}'_{i,1}, \hat{C}'_2$. Details can be found in [9].

4. If $(t'_{K-M+2}, \dots, t'_{K'-1})$ are dummies, the SP returns challenge $c \in \mathbb{Z}_p$.
5. The user computes and sends the following with q to the SP:

$$\begin{aligned} z_x &= r_x + cx, & z_s &= r_s + cs, & z_A &= r_t + cr_A, \\ z_e &= r_e + ce, & z_v &= r_v + cv, & z_{q'} &= r_{q'} + cq', \\ z_\beta &= r_\beta + cr_A e, & z_{y'} &= r_{y'} + cy', & z_y &= r_y + cy. \end{aligned}$$

For $i \in [1, K]$,

$$\begin{aligned} z_i &= r_i + ct_i, & z_{s_i} &= r_{s_i} + cs_i, & z_{y_i} &= r_{y_i} + cy_i, \\ z_{e_i} &= r_{e_i} + ce_i, & z_{\mu_i} &= r_{\mu_i} + c\mu_i, & z_{\beta_i} &= r_{\beta_i} + c\mu_i \beta_i. \end{aligned}$$

For $j \in \mathbb{J}$,

$$z_{v_j} = r_{v_j} + cv_j.$$

For $l \in [1, K-M]$,

$$\hat{z}_l = \hat{r}_l + c\hat{t}_l.$$

6. The SP checks if $T_1 A_2^c = h_1^{z_A}$, $T_2 = A_2^{-z_e} h_1^{z_\beta}$,

$$\begin{aligned}
 R\left(\frac{\hat{e}(A_1, w)}{\hat{e}(h_0, f)\hat{e}(h_2, f)^q}\right)^c &= \hat{e}(h_4, f)^{z_1}\hat{e}(h_5, f)^{z_2}\dots\hat{e}(h_{K+3}, f)^{z_K}\hat{e}(h_1, f)^{z_x}. \\
 &\hat{e}(h_3, f)^{z_s}\hat{e}(h_{K+4}, f)^{z_v}\hat{e}(\hat{h}, w)^{z_A}\hat{e}(\hat{h}, f)^{z_\beta}/\hat{e}(A_1, f)^{z_e}, \\
 R_{\mathbb{L}}\left(\frac{\hat{e}(\prod_{i=1}^K D_i^{\delta_i}, w_0)}{\hat{e}(h_0^{\sum_{i=1}^K \delta_i}, f)}\right)^c &= \hat{e}(h_1, f)^{\sum_{i=1}^K \delta_i z_i}\hat{e}(h_2, f)^{\sum_{i=1}^K \delta_i z_{s_i}} \cdot \hat{e}(h_3, f)^{\sum_{i=1}^K \delta_i z_{v_i}}. \\
 &\hat{e}(\hat{h}, w_0)^{\sum_{i=1}^K \delta_i z_{\mu_i}} \cdot \hat{e}(\hat{h}, f)^{\sum_{i=1}^K \delta_i z_{\beta_i}} / \hat{e}\left(\prod_{i=1}^K D_i^{\delta_i z_{e_i}}, f\right), \\
 R^* \cdot \hat{e}(V, w_2)^c &= \hat{e}(V, f_2)^{-z_s - \sum_{i=1}^K z_{s_i}} \cdot \hat{e}(g, f_2)^{z_v}, \\
 R'_M C'_M{}^c &= h_1^{z_x} h_2^{z_{q'}} h_3^{z_s + \sum_{j \in \mathbb{J}} z_{s_j}} \prod_{i=1}^{K-M+1} h_{i+3}^{\hat{z}_i} h_{K'+4}^{z_{y'}}, \\
 R_{\mathbb{J}} \cdot \hat{e}\left(\prod_{j \in \mathbb{J}} V_j^{\theta_j}, w_1\right)^c &= \hat{e}\left(\prod_{j \in \mathbb{J}} V_j^{-\theta_j z_j}, f\right) \hat{e}(g, f)^{\sum_{j \in \mathbb{J}} \theta_j z_{\alpha_j}}.
 \end{aligned}$$

It checks $\bar{T}_i B_i^c = h_1^{z_{\mu_i}}$, $\bar{W}_i = B_i^{-z_{e_i}} h_1^{z_{\beta_i}}$ for $i \in [1, K]$ and the zero-knowledge argument for the shuffling [9].

The SP computes: $A' = (h_0 C'_M h_{K-M+5}^{t'_{K-M+2}} \dots h_{K'+2}^{t'_{K'-1}} h_{K'+3}^{t'_{K'}} h_{K'+4}^{y''})^{\frac{1}{e'+\gamma}}$ for random $e', y'' \in \mathbb{Z}_p$, and sends (A', e', y'') to the user.

The SP then adds the entry $(t', 0, \sigma, \emptyset)$ to list \mathbb{T} , where $\sigma = (A, e, y)$ is the B-BBS+ signature with $A = (h_0 h_1^{t'} h_3^y)^{\frac{1}{\gamma+e}}$ for some random $e, y \in \mathbb{Z}_p$.

7. The user gets $\sigma'_A = (A', e', y' + y'')$ and updates its attributes to $A' = (x, q', s + s', \{t_1, \dots, t_{K-M+1}, t'_{K-M+2}, \dots, t'_{K'}\})$.

D Security

D.1 Simulation-Based Model

We use the simulation-based security definition following the literature [4, 45]. We consider a security game where an environment \mathcal{E} , which can schedule the invocation of the functionalities of SAC at its wish, is asked whether it is interacting with the real world or the ideal world. In the real world, all honest players communicate as specified in the protocol description. In the ideal world, the same players also follow the protocol except that they interact via a trusted party \mathcal{T} , responsible for handling all the inputs and outputs for them. The adversary \mathcal{A} in the real world takes control of some of the players and can communicate arbitrarily with environment \mathcal{E} . But \mathcal{A} does not know the communications between honest parties and the origin of messages received by \mathcal{A} .

Roughly, SAC is secure if, for any probabilistic polynomial time (PPT) algorithms \mathcal{A} and \mathcal{E} , there exists another algorithm \mathcal{S} , which has black-box access to \mathcal{A} , controlling the same players in the ideal world as \mathcal{A} does in the real world,

such that \mathcal{E} cannot distinguish if it is interacting with \mathcal{A} or \mathcal{S} . In other words, it also cannot distinguish between the real world and the ideal world. We first specify the functionalities of SAC in the real world and the ideal world, respectively:

1. **Setup.** The system starts when \mathcal{E} specifies the set of honest and dishonest users and the SP (static model).
 - *Real World.* The SP generates $(\mathbf{pp}, \mathbf{sk})$ and gives \mathbf{pp} to all players.
 - *Ideal World.* The trusted party \mathcal{T} initializes a database that stores the registration and authentication transcripts of all users. It also keeps track of the attributes of each user, and the public parameter \mathbf{pp} , which contains the status/score of each session.
2. **Registration.** \mathcal{E} asks user i to register with the SP.
 - *Real World.* User i registers with the SP, and both parties output individually the output of this interaction to \mathcal{E} . If user i has already been registered, the honest SP will reject the request. Similarly, an honest user discards the credential from the SP if it has successfully registered before.
 - *Ideal World.* User i sends a registration request to \mathcal{T} , who informs the SP about the request and whether user i has obtained a credential before. \mathcal{T} forwards the decision of the SP to user i . The user and the SP individually send the output of this interaction to \mathcal{E} . If the SP accepts the request and user i has not registered before, \mathcal{T} stores this transcript in its database.
3. **Authentication.** \mathcal{E} asks user i to authenticate and redeem some sessions.
 - *Real World.* User i authenticates with the SP w.r.t. the access policy f like a threshold score s_{th} and redeems the scores of sessions specified by \mathcal{E} . Both user i and the SP pass the local output of this interaction to \mathcal{E} .
 - *Ideal World.* User i sends an authentication request to \mathcal{T} , who checks according to \mathbf{pp} whether the user i satisfies the authentication condition. In more detail, \mathcal{T} maintains a database of the current and past tickets for each user, where the score of a user should match with the current \mathbf{pp} . \mathcal{T} informs the SP that some anonymous user wants to authenticate and whether the user satisfies the authentication condition. The SP replies with a new session identifier t or reject to \mathcal{T} , and \mathcal{T} forwards it to user i . If the authentication is successful, \mathcal{T} removes the redeemed sessions (\mathbb{T} specified by the user) from the user's attributes and updates its score. It also adds an entry for the active session t with score 0 to the database, and stores t as one of the user's session identifiers. The user and the SP individually send the output of this interaction and t (if not reject) to \mathcal{E} .
4. **Update.** \mathcal{E} asks the SP to update the score s to, or finalize, a session t .
 - *Real World.* The SP runs the update algorithm as instructed by \mathcal{E} .
 - *Ideal World.* \mathcal{T} updates its database accordingly as instructed by \mathcal{E} .

In the ideal world, all sessions are anonymous and unlinkable from the SP's view, and \mathcal{T} verifies whether the authenticating user satisfies the authentication condition. These capture completeness, anonymity, and soundness.

Definition 2. Let $\mathbf{Real}_{\mathcal{E},\mathcal{A}}(\lambda)$ (resp. $\mathbf{Ideal}_{\mathcal{E},\mathcal{S}}(\lambda)$) be the probability that \mathcal{E} outputs 1 when it runs in the real world (resp. ideal world) with adversary \mathcal{A} (resp. \mathcal{S} having black-box accesses to \mathcal{A}). SAC is secure if, for all PPT algorithms, $|\mathbf{Real}_{\mathcal{E},\mathcal{A}}(\lambda) - \mathbf{Ideal}_{\mathcal{E},\mathcal{S}}(\lambda)|$ is negligible in λ .

D.2 Proof

Theorem 2. Our scheme is secure if the BBS+ and weak BB signatures are existentially unforgeable, and the range proof, set membership proof, zero-knowledge argument of shuffling, \mathbb{P}_{Iss} , \mathbb{P}_{Sig} , \mathbb{P}_{Set} , and ZKPoK protocols are secure.

We describe how to construct \mathcal{S} in detail, except with the details of the underlying building blocks omitted (e.g., how to extract what \mathcal{S} needs from the ZKPoK). Firstly, \mathcal{S} maintains a list of credentials issued to \mathcal{A} during the life span of the system. \mathcal{S} also acts as an ideal-world adversary to the trusted party \mathcal{T} . \mathcal{S} simply forwards any messages between \mathcal{E} and \mathcal{A} . We consider two cases for \mathcal{S} :

Case 1: The SP is honest:

1. **Setup.** \mathcal{S} generates (pp, sk) and sends pp to \mathcal{A} .
2. **Registration.** \mathcal{S} acts as a dishonest user i (in the ideal world) to \mathcal{T} and an honest SP to \mathcal{A} as a dishonest user in the real world. Using the knowledge extractor of the ZKPoK protocol, \mathcal{S} extracts the value of x from \mathcal{A} . This value will be used to identify the dishonest user i . \mathcal{S} sends the request to \mathcal{T} on behalf of the user i . If \mathcal{T} replies accept, \mathcal{S} issues the credential to \mathcal{A} and also stores that credential.
3. **Authentication.** \mathcal{S} acts as a dishonest user i to \mathcal{T} and an honest SP to \mathcal{A} . \mathcal{S} extracts and uses the value x during authentication to determine user i . Two worlds are indistinguishable except in the rare events below:
 - During registration, \mathcal{S} fails to extract x from \mathcal{A} . This happens with negligible probability by the soundness property of \mathbb{P}_{Iss} .
 - During a successful authentication, \mathcal{S} fails to extract x from \mathcal{A} . This happens with negligible probability by the soundness property of \mathbb{P}_{Sig} .
 - There exists a successful authentication from \mathcal{A} such that \mathcal{S} outputs accept on behalf of an honest SP, but \mathcal{T} indicates the authenticating user does not satisfy the policy.

The last case implies that \mathcal{A} successfully did one of the following:

- forged a credential on attributes that has never been issued,
- obtained a credential on attributes with a ticket that is neither originated from the past version of the credential nor any dummy sessions,
- created one fake proof in authentication.

All these happen with negligible probability due to the following:

- BBS+ signatures are existentially unforgeable and \mathbb{P}_{Sig} is sound,
- the set membership proof is sound,
- the argument of shuffling is (computationally) sound, and
- the zero-knowledge proof is sound.

Since \mathcal{S} may need to run the extractor of the ZKPoK protocol, we require that the registration and authentications are run sequentially. A similar restriction also applies to PEREA, BLACR, and PERM.

Case 2: The SP is dishonest:

1. **Setup.** \mathcal{S} is given pp by \mathcal{A} .
2. **Registration.** \mathcal{S} acts as a dishonest SP to \mathcal{T} (in the ideal world) and an honest user i to \mathcal{A} . When \mathcal{T} requests registration for user i , \mathcal{S} runs the registration protocol with \mathcal{A} using the simulator of \mathbb{P}_{Iss} . If \mathcal{S} does not obtain a valid credential from \mathcal{A} , then \mathcal{S} replies *reject* to \mathcal{T} .
3. **Authentication.** \mathcal{S} acts as a dishonest SP to \mathcal{T} and an honest user to \mathcal{A} . When \mathcal{T} requests authentication for an anonymous user, \mathcal{S} runs the authentication protocol with \mathcal{A} . If \mathcal{T} proceeds and satisfies the authentication policy, \mathcal{S} uses the simulator of the ZKPoKs and shuffling protocol to simulate the view of \mathcal{A} using the random number q . If \mathcal{A} rejects, \mathcal{S} replies *reject* to \mathcal{T} .

The simulation provided to \mathcal{A} is correct due to the zero-knowledge property of the ZKPoK protocols and shuffling protocol and the hiding property of the commitment scheme. The behavior of \mathcal{S} in the ideal world is the same as that of \mathcal{A} in the real world.

References

1. Abe, M., Chow, S.S.M., Haralambiev, K., Ohkubo, M.: Double-trapdoor anonymous tags for traceable signatures. *Int. J. Inf. Secur.* **12**(1), 19–31 (2013)
2. Acar, T., Chow, S.S.M., Nguyen, L.: Accumulators and U-prove revocation. In: Sadeghi, A.-R. (ed.) *FC 2013. LNCS*, vol. 7859, pp. 189–196. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_15
3. Acar, T., Nguyen, L.: Revocation for delegatable anonymous credentials. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) *PKC 2011. LNCS*, vol. 6571, pp. 423–440. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19379-8_26
4. Au, M.H., Kapadia, A.: PERM: practical reputation-based blacklisting without TTPs. In: *CCS*, pp. 929–940 (2012)
5. Au, M.H., Kapadia, A., Susilo, W.: BLACR: TTP-free blacklistable anonymous credentials with reputation. In: *NDSS* (2012)
6. Au, M.H., Susilo, W., Mu, Y., Chow, S.S.M.: Constant-size dynamic k -times anonymous authentication. *IEEE Syst. J.* **7**(2), 249–261 (2013)
7. Backes, M., Hanzlik, L., Schneider-Bensch, J.: Membership privacy for fully dynamic group signatures. In: *CCS*, pp. 2181–2198 (2019)
8. Barki, A., Brunet, S., Desmoulins, N., Traoré, J.: Improved algebraic MACs and practical keyed-verification anonymous credentials. In: Avanzi, R., Heys, H. (eds.) *SAC 2016. LNCS*, vol. 10532, pp. 360–380. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_20
9. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) *EUROCRYPT 2012. LNCS*, vol. 7237, pp. 263–280. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_17
10. Bernstein, M.S., Monroy-Hernández, A., Harry, D., André, P., Panovich, K., Vargas, G.G.: 4chan and /b/: An analysis of anonymity and ephemerality in a large online community. In: *AAAI Conference on Web and Social Media (ICWSM)* (2011)

11. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptol.* **21**(2), 149–177 (2008)
12. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_3
13. Brickell, E., Li, J.: Enhanced Privacy ID: a direct anonymous attestation scheme with enhanced revocation capabilities. In: WPES, pp. 21–30 (2007)
14. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: S&P. pp. 315–334 (2018)
15. Camenisch, J., Chaabouni, R., Shelat, A.: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_15
16. Camenisch, J., Drijvers, M., Hajny, J.: Scalable revocation scheme for anonymous credentials based on n -times unlinkable proofs. In: WPES, pp. 123–133 (2016)
17. Camenisch, J., Kohlweiss, M., Soriente, C.: Solving revocation with efficient update of anonymous credentials. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 454–471. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15317-4_28
18. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_7
19. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_5
20. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0052252>
21. Chase, M., Meiklejohn, S., Zaverucha, G.: Algebraic MACs and keyed-verification anonymous credentials. In: CCS, pp. 1205–1216 (2014)
22. Chow, S.S.M.: Real traceable signatures. In: Jacobson, M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 92–107. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05445-7_6
23. Chow, S.S.M., Egger, C., Lai, R.W.F., Ronge, V., Woo, I.K.Y.: On sustainable ring-based anonymous systems. In: IEEE Computer Security Foundations (CSF) Symposium (2023, to appear)
24. Chow, S.S.M., Liu, J.K., Wong, D.S.: Robust receipt-free election system with ballot secrecy and verifiability. In: NDSS (2008)
25. Chow, S.S.M., Susilo, W., Yuen, T.H.: Escrowed linkability of ring signatures and its applications. In: Nguyen, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 175–192. Springer, Heidelberg (2006). https://doi.org/10.1007/11958239_12
26. Chow, S.S.M., Zhang, H., Zhang, T.: Real hidden identity-based signatures. In: Kiayias, A. (ed.) FC 2017. LNCS, vol. 10322, pp. 21–38. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70972-7_2
27. Couteau, G., Reichle, M.: Non-interactive keyed-verification anonymous credentials. In: Lin, D., Sako, K. (eds.) PKC 2019. LNCS, vol. 11442, pp. 66–96. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17253-4_3

28. Derler, D., Hanser, C., Slamanig, D.: A new approach to efficient revocable attribute-based anonymous credentials. In: Groth, J. (ed.) IMACC 2015. LNCS, vol. 9496, pp. 57–74. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-27239-9_4
29. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: the second-generation onion router. In: USENIX Security Symposium, pp. 303–320 (2004)
30. Doerner, J., Kondi, Y., Lee, E., abhi shelat, Tyner, L.: Threshold BBS+ signatures for distributed anonymous credential issuance. In: S&P. pp. 2095–2111 (2023)
31. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical short signature batch verification. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 309–324. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00862-7_21
32. Fiore, D., Garms, L., Kolonelos, D., Soriente, C., Tucker, I.: Ring signatures with user-controlled linkability. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds.) ESORICS Part II. LNCS, vol. 13555, pp. 405–426. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17146-8_20
33. Gurtler, S., Goldberg, I.: SoK: privacy-preserving reputation systems. *Proc. Priv. Enhancing Technol.* **2021**(1), 107–127 (2021)
34. Hajny, J., Dzurenda, P., Marques, R.C., Malina, L.: Privacy ABCs: now ready for your wallets! In: PerCom Workshops, pp. 686–691 (2021)
35. Jøsang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. *Decis. Support Syst.* **43**(2), 618–644 (2007)
36. Lai, R.W.F., Cheung, K., Chow, S.S.M., So, A.M.: Another look at anonymous communication. *IEEE Trans. Depend. Secur. Comput.* **16**(5), 731–742 (2019)
37. Lai, R.W.F., Ronge, V., Ruffing, T., Schröder, D., Thyagarajan, S.A.K., Wang, J.: Omniring: scaling private payments without trusted setup. In: CCS, pp. 31–48 (2019)
38. Lofgren, P., Hopper, N.: FAUST: efficient, TTP-free abuse prevention by anonymous whitelisting. In: WPES, pp. 125–130 (2011)
39. Ma, J.P.K., Chow, S.S.M.: SMART credentials in the multi-queue of slackness (or Secure management of anonymous reputation traits without global halting). In: IEEE European Symposium on Security and Privacy (EuroS&P) (2023, to appear)
40. Mittal, P., Olumofin, F.G., Troncoso, C., Borisov, N., Goldberg, I.: PIR-Tor: scalable anonymous communication using private information retrieval. In: USENIX Security Symposium (2011)
41. Narayanan, A., Shmatikov, V.: De-anonymizing social networks. In: IEEE Symposium on Security and Privacy, pp. 173–187 (2009)
42. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal verification of operations on dynamic sets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 91–110. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_6
43. Rosenberg, M., Maller, M., Miers, I.: SNARKBlock: federated anonymous blocklisting from hidden common input aggregate proofs. In: IEEE Symposium on Security and Privacy (S&P), pp. 1290–1307 (2022)
44. Tsang, P.P., Au, M.H., Kapadia, A., Smith, S.W.: Blacklistable anonymous credentials: Blocking misbehaving users without TTPs. In: CCS, pp. 72–81 (2007)
45. Tsang, P.P., Au, M.H., Kapadia, A., Smith, S.W.: PEREA: towards practical TTP-free revocation in anonymous authentication. In: CCS, pp. 333–344 (2008)
46. Xi, L., Feng, D.: FARB: fast anonymous reputation-based blacklisting without TTPs. In: WPES, pp. 139–148 (2014)

47. Yu, K.Y., Yuen, T.H., Chow, S.S.M., Yiu, S.M., Hui, L.C.K.: PE(AR)²: privacy-enhanced anonymous authentication with reputation and revocation. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 679–696. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33167-1_39
48. Zhang, T., Wu, H., Chow, S.S.M.: Structure-preserving certificateless encryption and its application. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 1–22. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12612-4_1



GeT a CAKE: Generic Transformations from Key Encapsulation Mechanisms to Password Authenticated Key Exchanges

Hugo Beguinet^{1,2(✉)}, Céline Chevalier^{1,3}, David Pointcheval¹,
Thomas Ricosset², and Mélissa Rossi⁴

¹ DIENS, École Normale Supérieure, CNRS, Inria, PSL University, Paris, France
celine.chevalier@ens.fr, david.pointcheval@ens.fr

² Thales, Gennevilliers, France

hugo.beguinet@ens.fr, hugo.beguinet@thalesgroup.com,
thomas.ricosset@thalesgroup.com

³ CRED, Université Paris-Panthéon-Assas, Paris, France

⁴ ANSSI, Paris, France

melissa.rossi@ssi.gouv.fr

Abstract. Password Authenticated Key Exchange (PAKE) have become a key building block in many security products as they provide interesting efficiency/security trade-offs. Indeed, a PAKE allows to dispense with the heavy public key infrastructures and its efficiency and portability make it well suited for applications such as Internet of Things or e-passports. With the emerging quantum threat and the effervescent development of post-quantum public key algorithms in the last five years, one would wonder how to modify existing password authenticated key exchange protocols that currently rely on Diffie-Hellman problems in order to include newly introduced and soon-to-be-standardized post-quantum key encapsulation mechanisms (KEM). A generic solution is desirable for maintaining modularity and adaptability with the many post-quantum KEM that have been introduced.

In this paper, we propose two new generic and natural constructions proven in the Universal Composability (UC) model to transform, in a black-box manner, a KEM into a PAKE with very limited performance overhead: one or two extra symmetric encryptions. Behind the simplicity of the designs, establishing security proofs in the UC model is actually non-trivial and requires some additional properties on the underlying KEM like fuzziness and anonymity. Luckily, post-quantum KEM protocols often enjoy these two extra properties. As a demonstration, we prove that it is possible to apply our transformations to Crystals-Kyber, a lattice-based post-quantum KEM that will soon be standardized by the National Institute of Standards and Technology (NIST).

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-33491-7_19.

In a nutshell, this work opens up the possibility to securely include post-quantum cryptography in PAKE-based real-world protocols.

Keywords: Key Encapsulation Mechanism · Password-Authenticated Key Exchange · Universal Composability

1 Introduction

A Password Authenticated Key Exchange (PAKE) protocol allows two users to derive a secret key over insecure channels only with the premise of sharing the same low entropy password. PAKE has become increasingly relevant in recent years due to the proliferation of connected devices and the growing demand for secure communication in scenarios where a public key infrastructure (PKI) may not be practical or desirable. It is particularly appealing for use cases like the Internet of Things (IoT) or e-passports, where portability, independence, and efficiency are important considerations. For IoT devices, for example, PKI is not feasible because of the number of devices and their limited computing resources and connectivity. PAKE allows these devices to securely communicate with each other using a simple password that can be easily changed if compromised. Similarly, in the case of e-passport, PAKE can be used to establish secure communication between the passport and a reader without the need for a PKI. It allows for a more portable and independent solution, as no central authority is necessary to verify a passport's authenticity. Overall, PAKE offers a trade-off between security and efficiency as compared to traditional authenticated key exchange protocols in certain circumstances.

Security models for PAKEs The security of PAKE will always be weaker than the security of PKI-based authenticated key exchange. Indeed, the presence of a low-entropy password allows powerful dictionary attacks. The conceptual idea in the PAKE security models is to accept the possibility of such dictionary attacks *but to prove that they must be made online*, i.e. that no password validity test is accessible offline. This slight security regression compared to authenticated key exchange is often accepted because online dictionary attacks are rarely relevant in practical contexts and the efficiency gain of PAKE is much higher. Moreover one can always block a user after a certain number of failed attempts. More formally, for proving the security of PAKE, the dictionary attacks should then be materialized in the existing security models for authenticated key exchange. Several solutions have emerged and have been refined over the last decade. Today, there are two main security models for PAKE protocols: the Bellare-Pointcheval-Rogaway [BPR00] and the Universal Composability (UC) model [Can01, CR03] with its PAKE's version [CHK+05]. The BPR model, introduced by Bellare, Pointcheval, and Rogaway, is a game-based security model that uses specific games to evaluate the ability of an adversary to break the protocol. On the other hand, the UC model, introduced by Canetti, is a simulation-based model that provides strictly better security guarantees, as stated in the original UC PAKE paper [CHK+05].

Existing work on PAKE. The concept of PAKE was formalized and analyzed during the 1990s by Bellare and Meritt with the Encrypted Key Exchange (EKE) protocol [BM92]. Since then, various PAKE protocols have been proposed, with some standardized by organizations such as the Internet Engineering Task Force (IETF) [Sch17]. Over the years, two main categories of PAKE appeared. The first use passwords to obscure the exchanged messages while the second use them as part of the randomness to build the necessary material, like the group generator. EKE [BM92] and OEKE [BCP03] are typical examples of the former. And SPEKE [Mac01] or CPace [AHH22] are examples for the latter. While many different PAKE designs have been introduced, not all are proven in the strong UC security model. In the previous examples of PAKE constructions, EKE [DHP+18], OEKE [ACCP08] and CPace [AHH21] have been proven in the UC model.

Post-quantum threat. While vastly used in current security products like IoT or e-passports, all these PAKE constructions rely on the Diffie-Hellman key exchange to provide cryptographic security. It raises concerns about their long-term security, as the emergence of quantum computing in recent years threatens any Diffie-Hellman-based key exchange, and thus any currently used PAKE. Indeed, quantum computers would potentially break, even retroactively, the mathematical foundations of many current cryptographic systems including the difficulty of the Diffie-Hellman problem. Therefore, it is crucial to carefully consider the long-term security of PAKE protocols and design them accordingly. In response to this potential threat to current cryptographic systems, the National Institute of Standards and Technology (NIST) has launched a standardization process for post-quantum cryptographic primitives in 2017. The goal of this campaign is to provide new post-quantum standards for two basic and crucial cryptographic building blocks: Key Encapsulation Mechanisms (KEMs) and digital signatures. These two families of public-key algorithms may be used on their own but more importantly, the future standards are destined to be included as black-boxes in internet and IoT protocols to complement the pre-quantum bricks. Many different families of mathematical problems were used for the design of candidate algorithms like error correcting codes or lattices. The analysis of the different candidate algorithms is currently ongoing but the NIST has announced a first set of standards in 2022 including the lattice-based KEM Crystals-Kyber [SAB+22]. More recently, specific PAKE constructions using post-quantum cryptography, in particular, lattices assumptions, were introduced. However most lattice constructions are either proven in weaker security models [GDLL17] or using mechanisms that are highly inefficient in practice [BCV19, ZY17].

1.1 Our Contributions

This paper proposes the first generic constructions to transform a black-box KEM into a PAKE. The idea is natural and inspired from EKE and OEKE. In high level, it consists in encrypting the public key using the password as a secret key. A second modification consists in either encrypting the ciphertext with that

same password or adding an authentication tag. The first transformation is called CAKE, derived from K(EM-to-P)AKE, and the second transformation is called OCAKE. Both constructions are graphically sketched later in the paper in Figs. 4 and 5. By design, they are simple, efficient and easy to implement. However, the price for such simplicity must be paid on the analysis side.

Let us first intuitively discuss the requirements on the KEM for achieving formal security. Consider a KEM where the public key is designed with a particular shape, for example, the public key might always be composed of small coefficients. When encrypted, the distribution of the sent message would look uniformly distributed. However, any attacker may perform an offline dictionary attack: given an encrypted public key, it will be possible to leverage the particular public key form as a condition for the valid password. In such case, the correct password will be the one that decrypts to a public key with small coefficients. Hence, an indistinguishability property on the distribution of the public key, called *fuzziness* (formally defined in Definition 3), will be required to avoid offline dictionary attacks. Likewise, such property on the ciphertext, called *anonymity* (formally defined in Definition 4), will be essential. In addition, another important property should be fulfilled by the symmetric encryption to construct such PAKE. The needed property is ensured by the Ideal Cipher (IC) model [BPR00]. It consists in assuming that the encryption behaves like a random permutation on every key. While it does not retain clear weaknesses to use a relaxed model, an ideal cipher is necessary to unwrap the proofs of our theorems.

In this paper, we successfully prove our CAKE and OCAKE constructions in the UC model assuming the above properties, the random oracle model (ROM) and the erasure model stating that any obsolete internal information is erased.

Why two constructions? Similarly to EKE and OEKE, CAKE and OCAKE offer slightly different security/efficiency trade-offs. Let us compare both constructions:

- The first construction, CAKE, consists in encrypting both exchanged messages. It leads to an implicitly authenticated key exchange protocol based on passwords. However a participant is not sure that the opposing party is able to obtain the session key. This assurance can only be achieved by explicit authentication. The security model for proving CAKE is very strong as it captures adaptive corruptions. In other words, the model allows an attacker to corrupt a user and thus obtain all its internal state in an adaptive way during an ongoing execution of the protocol.
- In order to add explicit authentication of the receiver, one usually includes a key-confirmation tag. In this case, one can remark with OCAKE that only one symmetric encryption is required. The second encryption is just replaced by the authentication that provides an explicit authentication to the receiver. However, this construction can only be proven secure in the static corruption model where the attacker may still corrupt users but the choice should be made before the execution of the protocol. Additionally, it is here possible to add an explicit client authentication at the end of the exchange to provide mutual explicit authentication.

In complement, we propose to show that the assumed properties on the KEM are not just artifacts that allow our proof to work. They are actually verified in concrete KEMs. We choose the example of Crystals-Kyber [SAB+22] as our guinea pig for applying our transformations. Crystals-Kyber is a future NIST post-quantum standard. We formally demonstrate that Kyber validates fuzziness and anonymity leading up to security statements for CAKE-Kyber (Theorem 3) and OCAKE-Kyber (Theorem 4).

1.2 Outline of the Paper

In Sect. 2, we introduce the preliminary notions on KEM, their security properties with some lattice definitions and a brief introduction to Kyber. In Sect. 3, we provide all the necessary information about PAKEs and their security in the UC model. In Sect. 4, we present both our KEM to PAKE transformations along with their security statements. For space reasons, the proofs are sketched in the main body of the paper and the full simulations are detailed in Appendices A and B (however full proofs are detailed in the same appendices of the full version that can be found here: [BCP+23]). Finally, we demonstrate our techniques on Kyber in Sect. 5.

2 Preliminaries

2.1 Notations

We note scalars, vectors, and matrices with lowercase plain (i.e. n), lowercase bold (i.e. \mathbf{e}), and uppercase bold (i.e. \mathbf{A}), respectively. We denote by $\text{negl}(\kappa)$ a negligible function of a security parameter κ . Given a finite set S , the notation $x \leftarrow_{\$} S$ means a uniformly random assignment of an element of S to the variable x . We note KEM the denomination of a key exchange mechanism and refer to KEM for the specific key encapsulation mechanism algorithm.

2.2 Key Encapsulation Mechanism

Even if the Key Encapsulation Mechanism's denomination is relatively recent, KEMs have been widely used throughout the history of public key cryptography. The first illustration is the fact that ElGamal [ElG85], based on the Diffie-Hellman key exchange, can be easily seen as a KEM. We will demonstrate later in this section that it enjoys several security notions, such as *semantic security*, *fuzziness*, and *anonymity*. Thereafter, with the NIST competition, many new researches have conducted to the introduction of KEM using a wide variety of structures. Let us cite a few examples: SABER [DKRV18,DKR+20], Crystals-Kyber [BDK+18,SAB+22], NewHope [ADPS16,PAA+19] on lattice-based assumptions or alternatively McEliece [McE78,ABC+22] on code-based assumptions.

Definition 1 (Key Encapsulation Mechanism). A Key Encapsulation Mechanism (KEM) is a triple of algorithms (*KeyGen*, *Encaps*, *Decaps*):

- *KeyGen*: Returns a pair of public-secret keys $(pk, sk) \in \mathcal{P} \times \mathcal{SK}$
- *Encaps*: Takes a public key $pk \in \mathcal{P}$ as input to produce a ciphertext $c \in \mathcal{C}$ and a key $K \in \mathcal{K}$. The ciphertext c is called an encapsulation of the key K ;
- *Decaps*: Takes a secret key $sk \in \mathcal{SK}$ and an encapsulation $c \in \mathcal{C}$ as input, and outputs $K \in \mathcal{K}$.

where \mathcal{SK} , \mathcal{P} , \mathcal{C} , and \mathcal{K} are the sets of secret keys, public keys, ciphertexts and session keys.

The formalization of the sets \mathcal{P} , \mathcal{C} , and \mathcal{K} will impact the security notions presented in the sequel.

Correctness. The correctness of a KEM requires that, for a security parameter κ ,

$$\Pr \left[\begin{array}{l} (pk, sk) \leftarrow \text{\$} \text{KeyGen}(1^\kappa) \\ (c, K) \leftarrow \text{Encaps}(pk) \end{array} : \text{Decaps}(sk, c) = K \right] > 1 - \text{negl}(\kappa).$$

Security Notions. The usual security notion for KEM is *semantic security*, also known as *indistinguishability*:

Definition 2 (Indistinguishability). We define the advantage of any adversary \mathcal{A} in deciding the key of the KEM by:

$$\text{Adv}_{\text{KEM}}^{\text{ind}}(\mathcal{A}) = \left| \Pr_{D_R}[\mathcal{A}(c, K) = 1] - \Pr_{D_S}[\mathcal{A}(c, K') = 1] \right|$$

where we consider the real and random distributions

$$\begin{aligned} D_R &= \{(pk, sk) \leftarrow \text{KeyGen}(1^\kappa); (c, K) \leftarrow \text{Encaps}(pk) : (c, K)\}, \\ D_S &= \{(pk, sk) \leftarrow \text{KeyGen}(1^\kappa); (c, K) \leftarrow \text{Encaps}(pk); K' \leftarrow \text{\$} \mathcal{K} : (c, K')\}. \end{aligned}$$

In all the advantage definitions, we will denote $\text{Adv}_{\text{KEM}}^{\text{ind}}(t)$ the maximal advantage any adversary can have within time t .

Let us introduce additional properties for KEMs, on the distributions of the public keys and of the encapsulations. We will denote by *fuzziness* the randomness of public keys, and by *anonymity* the randomness of the encapsulation. The latter is the usual definition, when the ciphertext distribution does not depend on the public key, and thus does not leak any information about the recipient.

Definition 3 (Fuzzy KEM). A KEM is said fuzzy if the distribution of the public keys output by the *KeyGen* algorithm are computationally indistinguishable from uniform keys in \mathcal{P} . More formally, we define the advantage of any adversary \mathcal{A} in breaking the fuzziness of the KEM by:

$$\text{Adv}_{\text{KEM}}^{\text{fuzzy}}(\mathcal{A}) = \left| \Pr_{D_R}[\mathcal{A}(pk) = 1] - \Pr_{D_S}[\mathcal{A}(pk) = 1] \right|,$$

where

$$D_R = \{(pk, sk) \leftarrow \text{KeyGen}(1^\kappa) : pk\} \quad \text{and} \quad D_{\mathfrak{s}} = \{pk \leftarrow_{\mathfrak{s}} \mathcal{P} : pk\}.$$

Definition 4 (Anonymous KEM). A KEM is said anonymous if the distribution of the ciphertexts outputted by the *Encaps* algorithm are computationally indistinguishable from uniform ciphertexts in \mathcal{C} . More formally, we define the advantage of any adversary \mathcal{A} in breaking the anonymity of the KEM by:

$$\text{Adv}_{\text{KEM}}^{\text{ano}}(\mathcal{A}) = \left| \Pr_{D_R}[\mathcal{A}(c) = 1] - \Pr_{D_{\mathfrak{s}}}[\mathcal{A}(c) = 1] \right|,$$

where

$$D_R = \{(pk, sk) \leftarrow \text{KeyGen}(1^\kappa); (c, K) \leftarrow \text{Encaps}(pk) : c\} \text{ and} \\ D_{\mathfrak{s}} = \{c \leftarrow_{\mathfrak{s}} \mathcal{C} : c\}.$$

ElGamal Key Encapsulation Mechanism. In order to illustrate the above notions, let us consider the particular KEM derived from the so-called ElGamal encryption scheme [ElG85]. Let \mathbb{G} be a group of prime order q , spanned by an element g :

- $\text{KeyGen}(1^\kappa)$: chooses a random $x \leftarrow_{\mathfrak{s}} \mathbb{Z}_q$ and sets $sk \leftarrow x$, $pk \leftarrow g^x$, with $\mathcal{SK} = \mathbb{Z}_q$ and $\mathcal{P} = \mathbb{G}$;
- $\text{Encaps}(pk)$: chooses a random $r \leftarrow_{\mathfrak{s}} \mathbb{Z}_q$ and sets $c \leftarrow g^r$, $K \leftarrow pk^r$, with $\mathcal{C} = \mathbb{G}$ and $\mathcal{K} = \mathbb{G}$;
- $\text{Decaps}(sk, c)$: outputs $K \leftarrow c^{sk}$.

It is well-know that the indistinguishability of this KEM relies on the Decisional Diffie-Hellman assumption. From the above description, this is clear that public keys are uniformly distributed in \mathbb{G} , hence this KEM is fuzzy; and the ciphertexts are also uniformly distributed in \mathbb{G} , thus this KEM is also anonymous. Note that the ElGamal KEM actually validates even stronger properties: perfect fuzziness (or smoothness) and perfect anonymity. The perfect nature comes from the fact that these notions are no longer computational but statistically ensured. As will be later stated in Remark 1, these properties are less common for post-quantum KEM protocols and thus will not be considered as requirements for our constructions.

2.3 Learning with Errors

Three rounds of the NIST standardization campaign are already over and one type of hardness assumption seems to be more enticing: lattices. Lattice problems provide strong worst-case to average-case reductions, making them excellent candidates for long-term security. Indeed state of the art algorithm using quantum adversaries are not far more efficient compared to current existing algorithm to solve LWE. In this section, we introduce the Learning With Errors (LWE) [Reg06] assumptions. It can be divided into two problems: a decisional

and a search problems. Both are assumed intractable in reasonable time, even for a quantum computer. Let us introduce the decisional version.

We directly consider this problem in a module structure named Module-LWE (we refer to [LS15] for more details). We define \mathcal{R}_q as the ring $\mathbb{Z}_q[X]/(X^n + 1)$. Let β_η be the distribution on \mathcal{R}_q where each coefficient of the polynomial is generated according to a centered binomial distribution with parameter 2η . We define the oracle $\mathcal{O}_{m,k,\eta}^{\text{mlwe}}$ that outputs samples of the form $(\mathbf{A}, \mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$ with $\mathbf{s} \leftarrow \beta_\eta^k$, $\mathbf{A} \leftarrow \mathcal{R}_q^{m \times k}$, and $\mathbf{e} \leftarrow \beta_\eta^m$.

Definition 5 (Decisional MLWE $_{m,k,\eta}$ Problem). *Given a set of parameters $m, k, \eta \in \mathbb{N}$, the advantage of any probabilistic polynomial time algorithm \mathcal{A} in deciding the d-MLWE over \mathcal{R}_q is:*

$$\text{Adv}_{m,k,\eta}^{\text{d-mlwe}}(\mathcal{A}) = \left| \frac{\Pr[(\mathbf{A}, \mathbf{b}) \leftarrow \mathcal{O}_{m,k,\eta}^{\text{mlwe}} : \mathcal{A}(\mathbf{A}, \mathbf{b}) = 1]}{-\Pr[(\mathbf{A}, \mathbf{b}) \leftarrow \mathcal{R}_q^{m \times k} \times \mathcal{R}_q^m : \mathcal{A}(\mathbf{A}, \mathbf{b}) = 1]} \right|.$$

2.4 CRYSTALS-Kyber

Crystals-Kyber, also known as Kyber, is a Module-LWE-based KEM that is one of the most efficient post-quantum solutions. It was introduced in response to the NIST call for standardization of post-quantum primitives and was accepted as the first post-quantum standard for key exchange in 2022. In its original paper [BDK+18], Kyber is proposed as a KEM that is secure against chosen-plaintext attacks (CPA-secure) and then achieves chosen-ciphertext attacks (CCA-secure) with the Fujisaki-Okamoto transform.

In Fig. 1, we present the CPA-secure version of Kyber, where \mathcal{R}_q is the ring $\mathbb{Z}_q[X]/(X^n + 1)$. Following the last supplemented version (3.0) [SAB+22], the suggested parameters are defined as follows: $(X^n + 1)$ is the $2n$ -th cyclotomic polynomial where n and q are equal respectively to 256 and $q = 3329$.

Additionally, for the sake of efficiency, Kyber includes an optimization using a compression function that can be thought of as a bit cut. While we do not use this compression in our protocol for the sake of clarity, it should be included in any implementation of our protocols using Kyber for maximum efficiency and correctness. We refer to the most recent NIST submission package [SAB+22] for more detailed information.

3 Password Authenticated Key Exchange

3.1 Introduction to PAKE

Initially introduced by Bellare and Merritt [BM92], a Password-Authenticated Key Exchange (PAKE) is a protocol that allows two parties to establish a shared secret session key over an insecure communication channel using a password as the only authentication means. The goal of PAKEs is to ensure that the key exchange is secure even if the password is weak or stolen by an attacker, the

Kyber.KeyGen(1^κ)	Kyber.Encaps($pk = (\rho, \mathbf{b})$)
1: $\rho, \sigma \leftarrow_{\$} \{0, 1\}^\kappa$	1: $\tau \leftarrow_{\$} \{0, 1\}^\kappa$
2: $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times k} := \text{Sam}(\rho)$	2: $m \leftarrow_{\$} \{0, 1\}^n \subseteq \mathcal{R}_q$
3: $(\mathbf{s}, \mathbf{e}) \leftarrow \beta_\eta^k \times \beta_\eta^k := \text{Sam}(\sigma)$	3: $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times k} := \text{Sam}(\rho)$
4: $\mathbf{b} \leftarrow \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$	4: $(\mathbf{r}, \mathbf{e}', \mathbf{e}'') \leftarrow \beta_\eta^k \times \beta_\eta^k \times \beta_\eta := \text{Sam}(\tau)$
5: return ($pk = (\rho, \mathbf{b}), sk = \mathbf{s}$)	5: $\mathbf{u} \leftarrow \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}'$
	6: $v \leftarrow \mathbf{b}^T \cdot \mathbf{r} + \mathbf{e}'' + \left\lceil \frac{q}{2} \right\rceil \cdot m$
	7: return $c \leftarrow (\mathbf{u}, v)$
Kyber.Decaps($sk = \mathbf{s}, c = (\mathbf{u}, v)$)	
1: return $\left\lceil \frac{2}{q} (v - \mathbf{s}^T \cdot \mathbf{u}) \right\rceil$	

Fig. 1. Simplified Kyber KEM: Kyber.KeyGen, Kyber.Encaps, Kyber.Decaps

only possible attack being an online exhaustive search, which can be detected and stopped using some organizational action.

PAKE protocols are handy when strong authentication is required while other forms of authentication (certificates) are not usable. They are often used in combination with other protocols to provide a secure channel for communication.

PAKE protocols might be vulnerable to two types of attacks: offline-dictionary attacks and online-dictionary attacks. The former occurs when an attacker gains knowledge of the password using pre computed lists of common passwords and exchanged information. Whereas, the latter involves an attacker actively trying to obtain the password by attempting to log in with different guesses. PAKE protocols often implement measures such as limiting the number of tries an attacker can make to guess the password to protect against online-dictionary attacks. Consequently, the security of a PAKE protocol ultimately relies on its resistance to offline-dictionary attacks. In other words, the strength of a PAKE protocol is determined by how difficult it is for an attacker to guess the password from the public transcript, even if it has a lot of time and resources.

3.2 The Universal Composability (UC) Model

Overview of the UC Framework. The Universal Composability (UC) model [Can01] is a simulation-based model in which an environment \mathcal{Z} attempts to differentiate the output of a protocol execution Π in the real world from the output generated in an ideal world. In the real world, the execution takes place between parties and an potential adversary. In the ideal world, dummy players and an ideal adversary or simulator \mathcal{S} interact solely with an ideal functionality \mathcal{F} to compute a specific function f . The ideal functionality can be informally defined as a trusted party that honestly and unconditionally responds to any

query. A schematic representation is given in Fig. 2.

The original paper [Can01] only uses *sid* as session identifiers, but a improved version of the UC model was published in [CR03] introducing sub-session identifiers *ssid*. For more clarity, throughout this article we use *ssid* for (*sid*, *ssid*). More explicitly, two *ssid* could theoretically be equal on different sessions *sid*, but by setting $ssid := (sid, ssid)$, we enforce the uniqueness of *ssid*. This uniqueness is necessary in the proofs provided in Appendices A and B.

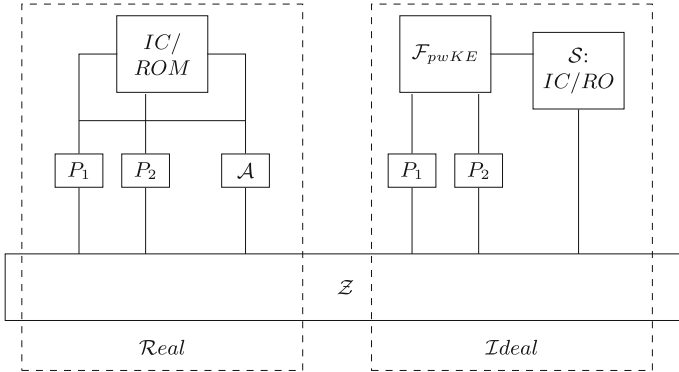


Fig. 2. *Real* versus *Ideal* world: \mathcal{Z} capability.

The goal of the UC model is to emulate the protocol Π using the ideal functionality. If the emulation is performed such that the environment \mathcal{Z} cannot distinguish (1) Π 's outputs with possible interactions with an adversary \mathcal{A} from (2) the outputs of dummy parties and a simulator interacting with the ideal functionality \mathcal{F} , then one can state that Π UC-emulates \mathcal{F} .

In our case, the protocols are Password-Based Authenticated Key Exchange (PAKE) and the ideal functionalities used throughout the paper specifically designed for PAKEs [CHK+05, ACCP08] are \mathcal{F}_{pwKE} and $\mathcal{F}_{pwKE-sA}$ (taken from [CHK+05] and defined in Fig. 3).

Ideal Functionality \mathcal{F}_{pwKE} . We present here the ideal functionality that is used for PAKEs. A detailed description of the functionality is given in [CHK+05]. It consists of three types of queries: **NewSession**, **TestPwd** and **NewKey**:

- **NewSession** allows a party to initialize a connection to another opposing party using its password. The functionality \mathcal{F}_{pwKE} uses this query to record the connection as well as the initial party's password.
- **TestPwd** models the unique online password test (online dictionary attacks) that is enabled through the execution of a PAKE. This query additionally impacts the view of the ideal functionality. Querying **TestPwd** changes the view of \mathcal{F}_{pwKE} in the exchange of two parties by altering the behavior of the next query **NewKey**, according to the correct or incorrect guess.

- **NewKey** interface allows to give parties a session key consistent with the state of their record. If two fresh entities do not share/use the same password then \mathcal{F}_{pwKE} does not give them the same key. Contrarily, if they do, this oracle returns the same key for both parties. However the behavior is more refined than that and takes account possible alterations from **TestPwd** (more details in Fig. 4).

Ideal Functionality with server authentication $\mathcal{F}_{pwKE-sA}$. We present a variation $\mathcal{F}_{pwKE-sA}$ of the previous ideal functionality to add explicit server authentication. A detailed description of the functionality is provided in Fig. 3.

- in each record in L (defined in Figure 3), we add a component $\text{role} \in \{\text{client}, \text{server}\}$. From this point forward, L has components of the form $(\text{ssid}, P_i, P_j, \text{pw}, \text{status}, \text{role})$.
- If the client queries **NewKey** at a time when the server still has not queried **NewKey** in the same session ssid , then $\mathcal{F}_{pwKE-sA}$ does nothing.
- If P_i and P_j do not share the same password: then the client gets **abort** whatever the status is.

Model. To prove that a protocol UC-emulates \mathcal{F}_{pwKE} or $\mathcal{F}_{pwKE-sA}$, we first need to set the model and assumptions for the proof. In this paper, we consider the Random Oracle Model (ROM) and the Ideal Cipher (IC) model (more details in full version [BCP+23]). We also make use of the erasure model and assume that the adversary \mathcal{A} is able to perform either adaptive or static corruptions, depending on the protocol.

Random Oracle. We use the definition of ROM introduced by Hofheinz and Müller-Quade [HM04]. This assumption provides a powerful tool that coherently responds to queries and generates answers that are uniformly random and independent of the input of the query.

Ideal Cipher. The ideal cipher model was first introduced in [BPR00]. It considers that a cipher behaves as a perfectly independent random permutation for every key used. We will generalize it a little bit by differentiating the input set and the output set, and then considering random bijections for every key.

Corruption. As mentioned earlier, we consider two types of corruptions in this paper: static corruptions and adaptive corruptions. Static corruptions allow the adversary to obtain the password of a party prior to the execution of the protocol. This means that during a simulation, the simulator knows which parties have been corrupted. Adaptive corruptions allow the adversary to corrupt any party during the execution of the protocol by revealing the password and internal state of the party.

Erasure model. The erasure model is a simple but powerful assumption. In this model, we assume that any internal information that is no longer useful ceases to exist. Therefore, in the event of information leakage, or adaptive corruption, previous internal information is not leaked as it no longer exists.

PAKE Ideal Functionality with server Authentication: $\mathcal{F}_{pwKE-sA}$ Session Initialization

On (**NewSession**, ssid, role, pw, P_i, P_j) from P_i :

- Sends (**NewSession**, ssid, role, P_i, P_j) to S .
- If this is the first **NewSession** query, or if it is the second **NewSession** query and there is a record $(\text{ssid}, P_j, P_i, \text{pw}', \star, \star) \in L$, then record $(\text{ssid}, P_i, P_j, \text{pw}, \text{fresh}, \text{role})$ in L .

Active attack

Upon receiving a query (**TestPwd**, ssid, P_i, pw') from the adversary \mathcal{S} , if there exists record $(\text{ssid}, P_i, P_j, \text{pw}, \text{fresh}, \text{role}) \in L$, do:

- If $\text{pw} = \text{pw}'$ mark the record as **compromised** and reply to \mathcal{S} with "*correct guess*".
- If $\text{pw} \neq \text{pw}'$, mark the record as **interrupted** and reply to \mathcal{S} with "*wrong guess*".

Key Generation

Upon receiving a query (**NewKey**, ssid, P_i, SK) from S , where $\text{SK} \in \{\text{keys}\}$, if there is a record of the form $(\text{ssid}, P_i, P_j, \text{pw}, \text{status}, \text{role}) \in L$, for any value **status**, and this is the first **NewKey** query for P_i :

- if role = client
 - If **status** = **compromised**, or if one of the players P_i or P_j is corrupted and there exists two records $(\text{ssid}, P_i, P_j, \text{pw}, \text{client})$ and $(\text{ssid}, P_j, P_i, \text{pw}, \text{server})$ then send (ssid, SK) to P_i ;
 - Else if **status** = **fresh** and there is a record $(\text{ssid}, P_j, P_i, \text{pw}', \text{client}')$ with $\text{pw}' = \text{pw}$, and a session key SK' has been sent to P_j , that was **fresh** at that time, then send $(\text{ssid}, \text{SK}')$ to P_i . Else if $\text{pw}' \neq \text{pw}$, choose a random key SK' whose length is k and send $(\text{ssid}, \text{SK}')$ to P_i .
 - Else if **status** = **fresh**, and if no record completed record exists for P_j in ssid do nothing.
 - Else if **status** = **interrupted**, send $(\text{ssid}, \text{error})$ to P_i .
- if role = server
 - If **status** = **compromised**, or if one of the players P_i or P_j is corrupted then send (ssid, SK) to P_i ;
 - Else if **status** = **fresh** or **status** = **interrupted**, choose a random key $\text{SK}' \in \{\text{keys}\}$ and send $(\text{ssid}, \text{SK}')$ to P_i .

Update the record as **completed**.

Fig. 3. $\mathcal{F}_{pwKE-sA}$: the ideal Functionality of a PAKE with server explicit authentication.

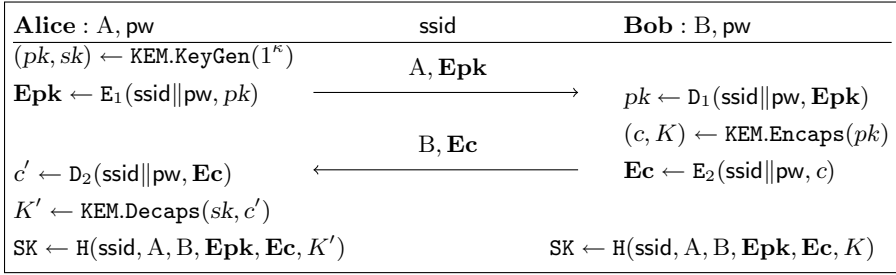


Fig. 4. CAKE with (E_1, D_1) , (E_2, D_2) two pairs of ideal ciphers. E_1 is a bijection from \mathcal{P} to \mathcal{P}' while E_2 is a bijection from \mathcal{C} to \mathcal{C}' .

4 Two Pieces of One Cake: Study of EKE and OEKE

In this study, we propose to examine the use of KEM with specific properties in the context of EKE and One-Way Encrypted Key Exchange (OEKE). We first introduce an evolved version of EKE called CAKE, that provides implicit authentication only, and then extend it to OEKE using a variant called OCAKE, that additionally provides explicit authentication of the receiver. We prove the security of these protocols in the Universal Composability (UC) model, assuming three properties of the KEM: semantic security, fuzziness, and anonymity. These two studies offer a balance between security properties and efficiency: CAKE handles adaptive corruptions, while OCAKE is proven secure in a relaxed model that only allows static corruptions to provide explicit authentication.

4.1 CAKE

In this subsection, we present a study of the K(EM)-EKE protocol, referred to as CAKE. This protocol is based on the use of generic KEM in EKE and is the most conservative of the two constructions we propose.

To study CAKE properly, as well as expressing the necessary properties on the underlying KEM, we first fix a KEM ($\text{KeyGen}, \text{Encaps}, \text{Decaps}$) with the sets \mathcal{SK} , \mathcal{P} , \mathcal{C} , and \mathcal{K} as in Definition 1. Next, we define two ideal cipher pairs (E_1, D_1) and (E_2, D_2) . We additionally define a set of keys Key and two sets \mathcal{P}' and \mathcal{C}' respectively bijections from \mathcal{P} and \mathcal{C} where both of them offer easy uniform sampling:

$$\begin{array}{ll}
 E_1 : \text{Key} \times \mathcal{P} \rightarrow \mathcal{P}' & E_2 : \text{Key} \times \mathcal{C} \rightarrow \mathcal{C}' \\
 D_1 : \text{Key} \times \mathcal{P}' \rightarrow \mathcal{P} & D_2 : \text{Key} \times \mathcal{C}' \rightarrow \mathcal{C}
 \end{array}$$

The actual keys of the ideal ciphers are the concatenations of the *ssid* and the passwords, to ensure independent bijections between different executions of the protocol. We introduce a description of CAKE in Fig. 4 along with its security theorem based on the fuzziness and anonymity of the underlying KEM in the ROM and IC models, while allowing adaptive corruptions.

Theorem 1. *Let (E_1, D_1) , (E_2, D_2) be two pairs of ideal ciphers and H be a random oracle. We note q_{D_1} (resp. q_{D_2}) the maximal number of queries to the decryption oracle D_1 (resp. D_2). We also note q_{E_1} (resp. q_{E_2}) the maximal number of queries to the encryption oracle E_1 (resp. E_2) explicitly asked by the adversary. Finally, we note q_s the number of sessions. The CAKE protocol described in Fig. 4 using KEM, a key encapsulation mechanism that is both fuzzy (Def. 3) and anonymous (Def. 4) ensuring semantic security, UC-emulates \mathcal{F}_{pwKE} in the erasure model with adaptive corruptions. More precisely, if we define $\text{Adv}_{\text{KEM}}^{\text{cake}}(\mathcal{A})$ the advantage of an adversary \mathcal{A} to break the above claim, it is bounded by*

$$\begin{aligned} & (2q_s + q_{D_1} + q_{D_2}) \cdot \text{Adv}_{\text{KEM}}^{\text{ind}}(t) \\ & + (q_s + q_{D_1}) \cdot \text{Adv}_{\text{KEM}}^{\text{fuzzy}}(t) + q_{D_1} \cdot (q_s + q_{D_2}) \cdot \text{Adv}_{\text{KEM}}^{\text{ano}}(t) \\ & + q_H \cdot q_s \cdot 2^{-\lambda_k} + q_{E_1}^2 \cdot 2^{-\lambda_p - 1} + q_{E_2}^2 \cdot 2^{-\lambda_c - 1}, \end{aligned}$$

where λ_k is the bit-length of the encapsulated keys, λ_p the bit-length of the public keys, and λ_c the bit-length of the ciphertexts, for the KEM scheme.

Sketch of Proof: In the subsequent games, we denote $\text{Pr}[\mathbf{G}]$ the probability for the environment \mathcal{Z} to output 1 in the simulated game \mathbf{G} . The goal is to prove that $\text{Pr}[\mathbf{G}]$ is close to the probability to output 1 in the ideal game, while starting from the real game \mathbf{G}_0 . The sequence of games will end with \mathbf{G}_9 that only uses the ideal functionality \mathcal{F}_{pwKE} , and is thus the ideal game. The complete simulation can be found in the Appendix A and the complete proof in the Appendix of the full version ([BCP+23]). We present here a sketch of proof:

- \mathbf{G}_0 : Real world protocol using the following assumptions: erasure model, random oracle, ideal cipher, adaptive corruption and lastly a fuzzy and anonymous KEM, which is also indistinguishable.
- \mathbf{G}_1 : Honest simulation of the random oracle H and the pairs of ideal ciphers (E_1, D_1) and (E_2, D_2) from Fig. 4, where we abort in case of collision during explicit encryption calls. Additionally, a private simulation of a random oracle H^* used for the simulation when \mathcal{S} cannot extract private information.
- \mathbf{G}_2 : Embedding of the secrets during the simulation of D_1 and D_2 .
- \mathbf{G}_3 : Simulation of Alice’s initialization with D_1 instead of E_1 .
- \mathbf{G}_4 : Simulation of Bob’s answer with D_2 instead of E_2 .
- \mathbf{G}_5 : Preparation of Alice’s reaction, by anticipating all the possible public keys decrypted by D_1 when a query is asked to D_2 .
- \mathbf{G}_6 : Simulation of Alice’s reaction, using the previous simulation of D_2 .
- \mathbf{G}_7 : Random session keys, where we replace all the unknown keys SK by random values.
- \mathbf{G}_8 : Adaptive corruptions, where we program the random oracle H and provide the secret values in case of corruption.
- \mathbf{G}_9 : Using on queries from \mathcal{F}_{pwKE} to detail the simulator in the ideal world.

A precise simulator is defined in Appendix A with Figs. 6, 7 and 8. □

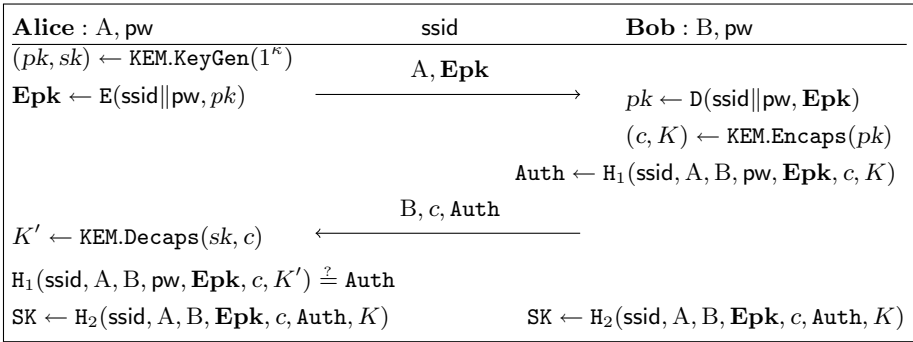


Fig. 5. OCAKE with (E, D) an ideal cipher. E is a bijection from \mathcal{P} to \mathcal{P}' .

Remark 1. Instantiated with the KEM derived from ElGamal presented in Sect. 2, CAKE-ElGamal is exactly the famous EKE [BM92]. But the proof technique actually differs because ElGamal enjoys perfect fuzziness (or smoothness) and perfect anonymity, which facilitate the EKE security proof. However, post-quantum algorithms cannot validate all these strong properties.

4.2 OCAKE

In this subsection, we modify the above CAKE protocol by adding explicit authentication of the receiver, which allows to remove one encryption. The modifications are based on the OEKE [BCP03] protocol but with generic KEM protocols instead of a Diffie-Hellman based key exchange.

In this setting, we only handle static corruptions in the security model. Indeed, it would have been possible to include adaptive corruptions in the security model with a statistical notion for the anonymity, i.e. perfect anonymity. But the computational property of our anonymity definition (see Definition 4) is more realistic for post-quantum KEM protocols. And thus here, adaptive corruptions cannot be handled by our proof in the UC-framework: in case of honest transcripts, we must generate a random c , on behalf of Bob. In case of Alice’s corruption, one can program E in order to set a specific (pk, sk) , but if the simulator commits on a specific c , then it cannot remain consistent. In particular, the adversary could have tried many passwords when decrypting \mathbf{Epk} , hence one cannot anticipate the public key for c .

In order to thoroughly study this approach, we outline the modifications in Fig. 5. We remove one encryption on the server flow to change it into an authentication. However for the sake of the proof, we have to slightly change how the hash query is usually done. Instead of using the public transcript, we use part of the secret information for the sake of the simulation in the security proof.

Lastly we use $\mathcal{F}_{pwKE-sA}$ in Fig. 3: the PAKE ideal functionality with explicit authentication of the server.

Theorem 2. *Let (E, D) be a pair of ideal cipher. Let H_1 and H_2 be two random oracles. We note q_D the maximal number of queries to the decryption oracle D . We also note q_E the maximal number of queries to the encryption oracle E , explicitly asked by the adversary. And we note q_s the number of sessions. The OCAKE protocol described in Fig. 5 using KEM, a key encapsulation mechanism that is both fuzzy (Def. 3) and anonymous (Def. 4) while ensuring semantic security, UC-emulates $\mathcal{F}_{pwKE-sA}$ in the erasure model with static corruptions. More precisely, if we define $\text{Adv}_{\text{KEM}}^{\text{ocake}}(\mathcal{A})$ the advantage of an adversary \mathcal{A} to break the above claim is bounded by*

$$(q_s + q_D) \cdot \text{Adv}_{\text{KEM}}^{\text{fuzzy}}(t) + (q_s + q_D + 1) \cdot \text{Adv}_{\text{KEM}}^{\text{ind}}(t) + q_D \cdot \text{Adv}_{\text{KEM}}^{\text{ano}}(\mathcal{A}) \\ + (q_{H_1} + 2q_s)^2 \cdot 2^{-\lambda_{H_1} - 1} + q_E^2 \cdot 2^{-\lambda_p - 1} + (q_{H_1} + q_{H_2}) \cdot q_s \cdot 2^{-\lambda_k},$$

where λ_k is the bit-length of the encapsulated keys, λ_p the bit-length of the public keys for the KEM scheme, and λ_{H_1} the bit-length of the authentication tag.

Sketch of Proof: Similarly to the proof of Theorem 1, in the subsequent games, we denote $\text{Pr}[\mathbf{G}]$ the probability for the environment \mathcal{Z} to output 1 in the simulated game \mathbf{G} . The goal is to prove that $\text{Pr}[\mathbf{G}]$ is close to the probability to output 1 in the ideal game, while starting from the real game \mathbf{G}_0 . The sequence of games will end with \mathbf{G}_8 that only uses the ideal functionality $\mathcal{F}_{pwKE-sA}$, and is thus the ideal game. The complete simulation can be found in the Appendix B and the complete proof is depicted in the same Appendix of the full version ([BCP+23]). We present here a sketch of proof:

- \mathbf{G}_0 : *Real world protocol using the following assumptions: erasure model, random oracle, ideal cipher, static corruption and lastly a fuzzy and anonymous KEM, which is also indistinguishable.*
- \mathbf{G}_1 : *Honest simulation of two random oracles H_1, H_2 and an ideal cipher (E, D) from Fig. 5, where we abort in case of collision during explicit encryption calls. Additionally a private simulation of each random oracle H_1^*, H_2^* used for the simulation when \mathcal{S} does not know any passwords. We also exclude collisions on H_1 and H_1^* .*
- \mathbf{G}_2 : *Embedding of the secret keys during the simulation of D .*
- \mathbf{G}_3 : *Simulation of an adversary finding Auth by chance.*
- \mathbf{G}_4 : *Simulation of Alice’s initialization with D instead of E .*
- \mathbf{G}_5 : *Simulation of Bob’s answer with (c, Auth) .*
- \mathbf{G}_6 : *Simulation of Alice’s reaction, using the Auth and the abortion in case the authentication is not verified.*
- \mathbf{G}_7 : *Random session keys, where we replace all the unknown authentication tags Auth and keys SK by random values, except for correctly guessed passwords*
- \mathbf{G}_8 : *Using on queries from $\mathcal{F}_{pwKE-sA}$ to detail the simulator in the ideal world.*

A precise simulator is defined in Appendix B with Figs. 9 and 10. □

Remark 2. Comparatively to remark 1, instantiated with the KEM derived from ElGamal presented in Sect. 2, OCAKE-ElGamal is exactly OEKE [ACCP08].

Additional remarks:

Removing the cipher on pk and keeping it on c like OEKE is not possible. To follow strictly its framework one would need a perfectly anonymous KEM otherwise the client could construct a subset attack using the gap of a miscon-structed cipher on decryption. Furthermore, we place our study in the con-text of quantum-secure KEM and none of them allows for perfect anonymity. Therefore the strict OEKE framework is not an enticing approach for long-term usability.

This protocol is only secure against static corruptions. An adversary allowed to apply adaptive corruptions could corrupt the client a reception of (c, Auth) before it computes Decaps . The adversary would then obtain sk because Decaps needs it and implies that the erasure is not applied. Knowing sk , a random $c \leftarrow \mathcal{C}$ is easily recognizable from a honestly built one leading the adversary to distinguish the simulation in the above proof.

Adding an authentication of the client afterwards for mutual authentication is entirely possible. The proof extensively use tricks before the derivation of the session key to either extract private information or to send indistinguishable random elements. Since this is done before, either the client would send a honest authentication, a perfectly indistinguishable one or a recognizable wrong one.

5 Crystal-Kyber

5.1 Security Properties

Crystals-Kyber has been introduced in Sect. 2. For the following results, we set $\mathcal{P} = \{0, 1\}^\kappa \times \mathcal{R}_q^k$, $\mathcal{SK} = \beta_\eta^k$, $\mathcal{C} = \mathcal{R}_q^k \times \mathcal{R}_q$ and $\mathcal{K} = \{0, 1\}^n$. First of all, we recall the indistinguishability property of Crystals-Kyber [BDK+18]:

Lemma 1. *Kyber on parameters (k, η, q, n) is an indistinguishable KEM:*

$$\text{Adv}_{\text{Kyber}}^{\text{ind}}(\mathcal{A}) \leq \text{Adv}_{k,k,\eta}^{\text{d-mlwe}}(t) + \text{Adv}_{k+1,k,\eta}^{\text{d-mlwe}}(t)$$

Next, let us verify the anonymity and fuzziness properties guaranteed by Kyber.

Lemma 2. *Crystals-Kyber is an anonymous KEM in $\mathcal{C} = \mathcal{R}_q^k \times \mathcal{R}_q$:*

$$\text{Adv}_{\text{Kyber}}^{\text{ano}}(\mathcal{A}) \leq \text{Adv}_{k,k,\eta}^{\text{d-mlwe}}(t) + \text{Adv}_{k+1,k,\eta}^{\text{d-mlwe}}(t)$$

Proof. Let sample a public key $pk \leftarrow \mathcal{S}(\mathbf{A}, \mathbf{b})$, by definition of `Kyber` in Fig. 1

$$c = (\mathbf{u}, v) \text{ with } \begin{cases} \mathbf{u} = \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}' \\ v = \mathbf{b}^T \cdot \mathbf{r} + e'' + \lceil \frac{q}{2} \rceil \cdot m \end{cases}$$

We can rewrite c as:

$$\begin{bmatrix} \mathbf{u} \\ v \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{A} \\ \mathbf{b} \end{bmatrix}^T \mathbf{r} + \begin{bmatrix} \mathbf{e}' \\ e'' + \lceil \frac{q}{2} \rceil \cdot m \end{bmatrix}$$

It forms a Module-LWE instance $\left(\begin{bmatrix} \mathbf{A} \\ \mathbf{b} \end{bmatrix}^T, \begin{bmatrix} \mathbf{u} \\ v \end{bmatrix} \right)$ provided that (\mathbf{A}, \mathbf{b}) is uniformly random on $\mathcal{R}_q^{k \times k} \times \mathcal{R}_q^k$ which is true under the $\mathbf{d}\text{-MLWE}_{k,k,\eta}$ assumption. This directly gives:

$$\text{Adv}_{\text{Kyber}}^{\text{ano}}(\mathcal{A}) \leq \text{Adv}_{k+1,k,\eta}^{\mathbf{d}\text{-mlwe}}(t) + \text{Adv}_{k,k,\eta}^{\mathbf{d}\text{-mlwe}}(t)$$

□

To prove anonymity in lemma 2, `Kyber` needs to ensure the decisional MLWE argument, therefore:

Corollary 1. *Crystals-Kyber is a fuzzy KEM with $\mathcal{P} = \mathcal{R}_q^k$:*

$$\text{Adv}_{\text{Kyber}}^{\text{fuzzy}}(\mathcal{A}) \leq \text{Adv}_{k,k,\eta}^{\mathbf{d}\text{-mlwe}}(t)$$

Theorem 3. *Let $(\mathbf{E}_1, \mathbf{D}_1), (\mathbf{E}_2, \mathbf{D}_2)$ be two pairs of ideal ciphers, and \mathbf{H} a random oracle. We note $q_{\mathbf{D}_1}$ (resp. $q_{\mathbf{D}_2}$) the maximal number of queries to the decryption oracle \mathbf{D}_1 (resp. \mathbf{D}_2), explicitly asked by the adversary, and q_s the number of session. The CAKE protocol from Fig. 4 instantiated with `Kyber` UC-emulates \mathcal{F}_{pwKE} in the erasure model with adaptive corruptions:*

$$\begin{aligned} \text{Adv}_{\text{Kyber}}^{\text{cake}}(\mathcal{A}) &\leq ((5q_s + 3q_{\mathbf{D}_1} + 2q_{\mathbf{D}_2}) + 2q_{\mathbf{D}_1} \cdot (q_s + q_{\mathbf{D}_2})) \cdot \text{Adv}_{k+1,k,\eta}^{\mathbf{d}\text{-mlwe}}(t) \\ &\quad + q_{\mathbf{H}} \cdot q_s \cdot 2^{-n} + q^{-kn} \cdot (q_{\mathbf{E}_1}^2 \cdot 2^{-\kappa} + q_{\mathbf{E}_2}^2 \cdot q^{-n})/2 \end{aligned}$$

Theorem 4. *Let (\mathbf{E}, \mathbf{D}) be an ideal cipher, and $\mathbf{H}_1, \mathbf{H}_2$ two random oracles. We note $q_{\mathbf{D}}$ the maximal number of queries to the decryption oracle \mathbf{D} , explicitly asked by the adversary, and q_s the number of session. The OCAKE protocol from Fig. 5 instantiated with `Kyber` UC-emulates $\mathcal{F}_{pwKE-sA}$ in the erasure model with static corruptions:*

$$\begin{aligned} \text{Adv}_{\text{Kyber}}^{\text{ocake}}(\mathcal{A}) &\leq 2 \cdot q_{\mathbf{D}} \cdot (\text{Adv}_{k+1,k,\eta}^{\mathbf{d}\text{-mlwe}}(t)) + 3 \cdot q_{\mathbf{D}} \cdot (\text{Adv}_{k,k,\eta}^{\mathbf{d}\text{-mlwe}}(t)) \\ &\quad + (q_{\mathbf{H}_1} + q_{\mathbf{H}_2}) \cdot q_s \cdot 2^{-n} + q_{\mathbf{E}}^2 \cdot 2^{-\kappa} + q_{\mathbf{H}_1} \cdot 2^{-n} \end{aligned}$$

5.2 Instantiation of the Block Cipher

To prove the UC-security of both CAKE-Kyber and OCAKE-Kyber, the ideal cipher model is crucial. More precisely it needs to ensure that finding a collision on the encryption is statistically impossible without querying a decryption oracle. It removes both the following approaches out of the equation: stream cipher and one time pad. The conception of a relevant block cipher for our transformations is actually nontrivial. In fact, the underlying sets, like \mathcal{R}_q , are not convenient for building a symmetric block cipher statistically following the necessary ideal properties. We present here a solution issued from known ad-hoc techniques. We believe that it can be improved for better performance but this task is left as future work.

To keep light notations, we do not encrypt the seed of \mathbf{A} , and thus consider the encryption of an element in $\mathcal{R}_q^k \sim \mathbb{Z}_q^{n \times k}$ for the public key. We can do the same with $\mathcal{R}_q^k \times \mathcal{R}_q \sim \mathbb{Z}_q^{n \times (k+1)}$ for the ciphertext.

To encrypt $pk \in \mathcal{R}_q^k \sim \mathbb{Z}_q^{n \times k}$, we can first encode pk into $\{0, \dots, q^{nk} - 1\}$, and then use a block cipher on ℓ -bits, such that $2^{\ell-1} \leq q^{nk} < 2^\ell$. We thus have an encoding/decoding from \mathcal{R}_q^k to $\{0, 1\}^\ell$ that can be seen as a superset of $\{0, \dots, q^{nk} - 1\}$. Let us thus consider all these encodings equivalent.

From (E, D) on ℓ -bit blocks and κ -bit keys, we can build a permutation onto the restricted set $\{0, \dots, q^{nk} - 1\}$: one defines the encryption scheme with key K on $\mathbf{b} \in \mathcal{R}_q^k \sim \{0, \dots, q^{nk} - 1\}$ as $E'_K(\mathbf{b}) = E_K(\dots E_K(\mathbf{b}) \dots)$, stopping at the first element in $\{0, \dots, q^{nk} - 1\}$. Decryption works the same way, and will stop at the right place as all ignored intermediate values are outside the expected set.

Actually, the number of iterations will be small, as there is a probability less than $1/2$ at each step. This technique is vulnerable to timing attacks, but one can always include virtual loops.

We emphasize that this study is done without using the optimized **Kyber** (without) the **compression**, **decompression** functions. However, these two functions map elements of \mathbb{Z}_q to $\mathbb{Z}_{q'}$ with $q' < q$, therefore the study remains similar.

5.3 Parameters

The bounds in Theorems 3 and 4 are slightly looser than the ones that constrain the choice of parameters for **Kyber**. Thus, some adaptations of the obtained security levels are necessary. We propose to recompute the security level for the set of parameters taken from **Kyber**'s last submission to the NIST [SAB+22] and choosing parameters that allow to reach around 100 bits of security against quantum adversaries. While this can be argued to be weak, **PAKE** are not used in highly critical applications but in highly efficient ones. Hence, 100 bits of security against quantum adversaries would constitute a mid to long-term security target.

We present in Table 1 the security estimations for CAKE-Kyber and OCAKE-Kyber obtained with the pq-crystal estimate [DS21] against a quantum adversary with **KYBER768** and **KYBER1024** parameters.

Table 1. Bit security estimates of CAKE-Kyber and OCAKE-Kyber using parameters from version 3.0 of the NIST [SAB+22] against a quantum adversary. Estimation done using python script from pqcrystals github [DS21].

Kyber parameters	Bit-sec against quantum adversaries obtained with [DS21]
Kyber1024	102

CAKE-Kyber

Kyber768	98
Kyber1024	162

OCAKE-Kyber

The security/efficiency trade-off between CAKE and OCAKE is confirmed in this example. According to Table 1, CAKE provides more conservative assumptions as an adaptive adversary are included in the model however it is less efficient than its OCAKE alternative.

We emphasize the fact that this paper propose a generic construction proven secure in the UC model. To our knowledge before this work no UC-secure lattice-based PAKE are usable in practice. However BPR-secure PAKE such as [GDLL17] should be slightly more efficient in practice due to our reduction. One benefit of our approach is that the efficient and secure implementation of Kyber can directly be applied.

6 Conclusion and Perspectives

In this article we characterize the necessary properties for a key encapsulation mechanism to be used in a password authenticated key exchange and more precisely in both EKE and OEKE. Additionally we prove that these properties are respected by the newly standardized Kyber. To supplement this study we introduce a set of possible parameters for Kyber, ensuring around 100 bit of security. Lastly we propose a cipher respecting statistically ideal cipher properties for the application of both CAKE (Fig. 4) and (OCAKE Fig. 5).

While our work focuses on post-quantum alternatives, one could improve our results by supposing a random self-reducible KEM (like El-Gamal). Random self-reducibility implies that arbitrarily many independant instances can be reduced

to only one such instance. Although current post-quantum schemes are not self-reducible KEM but such an assumption would lead to tighter reductions and it would allow for SPEKE or CPace constructions to be generalized to more KEM protocols.

Acknowledgements. We would like to thank Olivier Blazy and Henri Gilbert respectively for useful discussions about password-authenticated key exchange protocols and their security and symmetric encryption for the PAKE practicability. This work was supported in part by the French Programme d'Investissement d'Avenir (PIA) under national project RESQUE and by the French ANR projects CryptiQ (ANR-18-CE39-0015) and SecNISQ (ANR-21-CE47-0014). The first author was also supported by ANRT under the program CIFRE N° 2021/0645.

References

- ABC+22. Albrecht, M.R., et al.: Classic McEliece. Technical report, National Institute of Standards and Technology (2022). <https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions>
- ACCP08. Abdalla, M., Catalano, D., Chevalier, C., Pointcheval, D.: Efficient two-party password-based key exchange protocols in the UC framework. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 335–351. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79263-5_22
- ADPS16. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: NewHope without reconciliation. Cryptology ePrint Archive, Report 2016/1157 (2016). <https://eprint.iacr.org/2016/1157>
- AHH21. Abdalla, M., Haase, B., Hesse, J.: Security analysis of CPace. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. Part IV, volume 13093 of LNCS, pp. 711–741. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-92068-5_24
- AHH22. Abdalla, M., Haase, B., Hesse, J.: CPace, a balanced composable PAKE. Internet-Draft draft-irtf-cfrg-pace-06, Internet Engineering Task Force. Work in Progress, July (2022)
- BCP03. Bresson, E., Chevassut, O., Pointcheval, D.: Security proofs for an efficient password-based key exchange. In: Jajodia, S., Atluri, V., Jaeger, T., editors, ACM CCS 2003, pp. 241–250. ACM Press, October (2003)
- BCP+23. Beguinet, H., Chevalier, C., Pointcheval, D., Ricosset, T., Rossi, M.: Get a cake: generic transformations from key encapsulation mechanisms to password authenticated key exchanges. Cryptology ePrint Archive, Paper 2023/470 (2023). <https://eprint.iacr.org/2023/470>
- BCV19. Blazy, O., Chevalier, C., Huy Vu, Q.: Post-quantum uc-secure oblivious transfer in the standard model with adaptive corruptions. In: Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26–29, 2019, pp. 28:1–28:6. ACM (2019)
- BDK+18. Bos, J.W.: CRYSTALS - kyber: a cca-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24–26, 2018, pp. 353–367. IEEE (2018)
- BM92. Bellare, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy, pp. 72–84. IEEE Computer Society Press, May (1992)


- BPR00. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_11
- Can01. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October (2001)
- CHK+05. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_24
- CR03. Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (2003)
- DHP+18. Dupont, P.-A., Hesse, J., Pointcheval, D., Reyzin, L., Yakubov, S.: Fuzzy password-authenticated key exchange. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 393–424. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_13
- DKR+20. D’Anvers, J.-P. et al.: SABER. Technical report, National Institute of Standards and Technology (2020). <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>
- DKRV18. D’Anvers, J.-P., Karmakar, A., Sinha Roy, S., Vercauteren, F.: Saber: module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2018. LNCS, vol. 10831, pp. 282–305. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89339-6_16
- DS21. Ducas, L., Schanck, J.: pq-crystals/security-estimates. <https://github.com/pq-crystals/security-estimates> (2021)
- ElG85. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**, 469–472 (1985)
- GDLL17. Gao, X., Ding, J., Liu, J., Li, L.: Post-quantum secure remote password protocol from RLWE problem. Cryptology ePrint Archive, Report 2017/1196 (2017). <https://eprint.iacr.org/2017/1196>
- HM04. Hofheinz, D., Müller-Quade, J.: Universally composable commitments using random oracles. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 58–76. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_4
- LS15. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Des. Codes Cryptogr. **75**(3), 565–599 (2015)
- Mac01. MacKenzie, P.: On the security of the SPEKE password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057 (2001). <https://eprint.iacr.org/2001/057>
- McE78. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. The deep space network progress report 42–44, Jet Propulsion Laboratory, California Institute of Technology, January/February (1978). https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF
- PAA+19. Poppelmann, T., et al.: NewHope. Technical report, National Institute of Standards and Technology (2019). <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>
- Reg06. Regev, O.: Lattice-based cryptography. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 131–141. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_8

- SAB+22. Schwabe, P., et al.: CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology (2022). <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>
- Sch17. Schmidt, J.-M.: Requirements for password-authenticated key agreement (PAKE) schemes. RFC **8125**, 1–10 (2017)
- ZY17. Zhang, J., Yu, Yu.: Two-round PAKE from approximate SPH and instantiations from lattices. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. Part III, volume 10626 of LNCS, pp. 37–67. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-70700-6_2

Multiparty Computation



Explicit and Nearly Tight Lower Bound for 2-Party Perfectly Secure FSS

Keitaro Hiwatashi^{1,3} and Koji Nuida^{2,3}

¹ Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan

keitaro_hiwatashi@mist.i.u-tokyo.ac.jp

² Institute of Mathematics for Industry, Kyushu University, Fukuoka, Japan
nuida@imi.kyushu-u.ac.jp

³ National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

Abstract. Function Secret Sharing (FSS) is a cryptographic tool introduced by Boyle et al. (EUROCRYPT 2015) and is useful for several applications such as private information retrieval, oblivious-RAM, multi-party computation, etc. Most of the known FSS schemes are based on a pseudorandom generator and hence with computational security. In contrast, there are only a few known constructions of information-theoretic FSS, which are just for restricted function classes. It has not been well studied how efficient information-theoretic FSS can be in general. In this paper, we focus on (2-party) perfectly secure information-theoretic FSS and prove that the key size is explicitly (i.e., not just asymptotically) bounded below by the size of the subgroup generated by the function class. To the best of our knowledge, this is the first lower bound for information-theoretic FSS for an arbitrary function class. Our result shows that for several practically meaningful function classes, perfectly secure information-theoretic FSS must be much inefficient, not only asymptotically but also in practical parameters. Furthermore, we prove that this explicit lower bound is nearly tight by constructing perfectly secure information-theoretic FSS schemes for arbitrary function classes almost achieving our lower bound.

Keywords: function secret sharing · lower bounds · information-theoretic security

1 Introduction

Function secret sharing (FSS), introduced by Boyle et al. [2], is informally a secret sharing scheme on a function class \mathcal{F} . That is, (2-party) FSS splits a function $f \in \mathcal{F}$ into two functions f_0 and f_1 , which are called keys of f , such that (1) $f(x) = f_0(x) + f_1(x)$ holds for all inputs x and (2) each f_b does not reveal any information of f . The splitting algorithm is called the key generation algorithm and the algorithm to compute $f_b(x)$ is called the evaluation algorithm. Since the introduction of FSS, many applications have been proposed: e.g., distributed

oblivious RAM [5–7] from FSS for point functions; private queries (including private information retrieval, PIR) [10, 12] from FSS for point functions and interval functions; metadata-hiding messaging [8] from FSS for point functions; and multi-party computation (MPC) [1, 4] from FSS for point functions, interval functions, bit-decomposition functions, etc.

Most of the known FSS schemes [1–6] are based on a pseudorandom generator and hence with computational security. In contrast, there are only a few perfectly secure information-theoretic FSS (IT-FSS) schemes [4]. One may wonder why there are only a few IT-FSS, while most of the (ordinary) secret sharing schemes are perfectly secure. More specifically, in (ordinary) secret sharing, we already know an efficient construction of a perfectly secure secret sharing scheme (for threshold access structures) for a set S with only $\log |S|$ bits share size. Coming back to the case of FSS, is it also possible to construct an IT-FSS scheme for a function class \mathcal{F} with $\log |\mathcal{F}|$ bits key size?

Unfortunately, two negative results are known regarding efficient IT-FSS:

1. Gilboa and Ishai [9] stated that the information-theoretic distributed point functions (DPF), which is a special case of FSS for point functions, need $\Omega(2^n)$ bits key size, where n is the bit size of input space. Furthermore, they also showed that if there exists an efficient DPF scheme, then a one-way function exists.
2. Boyle et al. [2] proved that if there exists an efficient FSS scheme for a *poly-spanning* function class, which includes point functions and interval functions, then a pseudorandom function exists.

However, their results do not cover some function classes used in applications: For example, their results showed that FSS for a function class $\{f_{\alpha,\beta}\}$ over $\alpha \in \mathbb{Z}_{2^n}$ and $\beta \in \mathbb{Z}_{2^n}$, where $f_{\alpha,\beta}$ is a function such that $f_{\alpha,\beta}(\alpha) = \beta$ and $f_{\alpha,\beta}(x) = 0$ for $x \neq \alpha$, implies the existence of one-way functions. (Therefore, it seems difficult to construct an IT-FSS scheme for the function class.) However, an FSS scheme for its subclass $\{f_{\alpha,1}\}$ over $\alpha \in \mathbb{Z}_{2^n}$ does not imply the existence of one-way functions by their argument, since the function class is not poly-spanning. (Therefore, there is a possibility to construct an efficient IT-FSS scheme for this function class.) This function class is used in the equality test in MPC [4]. See Sect. 3 for more details.

Additionally, the two negative results are based on asymptotic analysis. However, in applications, non-asymptotic efficiency evaluation on practical parameters is also important. There is a possibility that in practical parameters, an IT-FSS scheme with an asymptotically large key size outperforms an FSS scheme based on pseudorandom generators with an asymptotically small key size.

In the context of the above, we tackle the question of how much we can improve the key size of IT-FSS.

1.1 Our Contributions

We give an explicit lower bound for the key size of (2-party) IT-FSS for an arbitrary function class. To the best of our knowledge, it is the first lower bound

for IT-FSS for an arbitrary function class. Also, our lower bound is explicit and therefore is useful for analyzing the efficiency in fixed practical parameters, while the known lower bound for IT-FSS for point functions is just asymptotic. Using our explicit lower bound, we can clarify that the key size of an IT-FSS scheme is much larger than that of an FSS scheme based on pseudorandom generators, even in practical parameters. For example, our lower bound implies the key size of an IT-FSS scheme for point functions with $n = 64$ bits input and output space is at least $2^{70} - 65$ bits, while the key size of the computationally secure scheme in [3] for the same function class is 8576 bits.

Furthermore, we prove that this explicit lower bound is nearly tight by constructing IT-FSS schemes for arbitrary function classes almost achieving our lower bound. For a general case, the gap in the key size of our construction compared to our lower bound is only almost the bit size of the range of functions. The gap becomes smaller for some special cases: especially, when the range of functions is an elementary Abelian 2-group, the key size of our construction is just one bit larger than our lower bound.

1.2 Technical Overview

We provide an overview of our techniques. We explain more details in the following sections.

Explicit Lower Bound. We give an explicit lower bound for IT-FSS for a function class \mathcal{F} in two steps:

1. We convert an IT-FSS scheme to a *simplified* IT-FSS scheme with (at most) 1-bit additional key size (Theorem 2).
2. We give an explicit lower bound for a simplified IT-FSS scheme (Corollary 1 and Corollary 2).

Informally, a *simplified* IT-FSS scheme is a symmetric and non-redundant IT-FSS scheme. “Symmetric” means that the evaluation algorithm is independent of party id. “Non-redundant” means that the behavior of the evaluation algorithm differs when the key is different. We can convert an IT-FSS scheme to a symmetric IT-FSS scheme simply by embedding a party id into a key. That is, key generation algorithm outputs $(b||k_b, (1 \oplus b)||k_{1 \oplus b})$ ¹ as a key pair, where (k_0, k_1) is a key pair generated in the original key generation algorithm and b is a random bit. The evaluation algorithm parses the key into the form of $b||k_b$ and invokes the based evaluation algorithm as a party b . Then we convert a symmetric IT-FSS scheme to a simplified IT-FSS scheme. This can be realized by identifying keys of which the behavior of the evaluation algorithm is the same.

In the second step, we give an explicit lower bound for a simplified IT-FSS scheme. First, due to the non-redundant property for the keys, we show that the possible keys for the first party are in one-to-one correspondence to the possible

¹ “||” denotes a concatenation.

keys for the second party. That is, given a key k_0 , which is one component of a key pair, and a function f , the other component k_1 is uniquely determined. We express this mapping as $k_0 \xrightarrow{f} k_1$. Using this mapping sequentially, a sequence of functions (f_0, \dots, f_{n-1}) corresponds to a sequence of keys (k_0, \dots, k_n) as follows:

$$k_0 \xrightarrow{f_0} k_1 \xrightarrow{f_1} \dots \xrightarrow{f_{n-1}} k_n.$$

Then from the correctness, i.e., the fact that the sum of outputs of the evaluation algorithm is equal to $f(x)$, there exists an algebraic relationship between a sequence of functions and the corresponding sequence of keys. Using this algebraic relationship, we can construct an injection from \mathcal{F}' to the key space, where \mathcal{F}' is a certain set determined by \mathcal{F} . Therefore, the key size is explicitly bounded below by the size of \mathcal{F}' .

Nearly Optimal Construction. Our construction is similar to a one-time truth table [4, 11], which can be seen as an additive secret sharing on a set of all functions. In the case that the function class \mathcal{F} is closed under addition, we can use an ordinary additive secret sharing scheme to generate a “share” (regarded as a key) of a function. Therefore, we can construct an IT-FSS scheme for function class $\langle \mathcal{F} \rangle$, where $\langle \mathcal{F} \rangle$ is the linear span of \mathcal{F} , with $\log |\langle \mathcal{F} \rangle|$ bits key. Of course, this is also an IT-FSS scheme for function class $\mathcal{F} \subseteq \langle \mathcal{F} \rangle$. Surprisingly, from a simple argument using group theory, we can show that this construction almost achieves our lower bound.

1.3 Organization

We provide the notations used in this paper and the definition of information-theoretic FSS in Sect. 2. In Sect. 3, we review the known negative results regarding IT-FSS. We show an explicit lower bound for an IT-FSS scheme for an arbitrary function class in Sect. 4. In Sect. 5, we give an IT-FSS scheme that nearly achieves the lower bound given in Sect. 4.

2 Preliminaries

In this section, we review the basic notations used in this paper (Sect. 2.1) and the definition of FSS considered in this paper (Sect. 2.2).

2.1 Notation

We let $\log x$ denote the logarithm of x to base 2, $s||s'$ denote the concatenation of two strings s and s' , and $\pi \circ \sigma$ for two permutations π and σ denote the composed permutation, i.e., $\pi \circ \sigma(x) = \pi(\sigma(x))$ for all x . For a set \mathcal{X} , $|\mathcal{X}|$ denotes the size of \mathcal{X} . For an integer n , $[n]$ denotes a set $\{0, 1, \dots, n - 1\}$. We let \mathbb{G} denote an Abelian group, $+$ denote the operation on \mathbb{G} , and 0 denote the identity element

of \mathbb{G} . For $g \in \mathbb{G}$, $-g$ denotes the inverse element of g , and we write $g - g'$ instead of $g + (-g')$. For a set \mathcal{X} , $\mathbb{G}^{\mathcal{X}}$ denotes the set of all functions $\mathcal{X} \rightarrow \mathbb{G}$. $\mathbb{G}^{\mathcal{X}}$ can be seen as a direct product $\prod_{i=1}^{|\mathcal{X}|} \mathbb{G}$ and the operation of $\mathbb{G}^{\mathcal{X}}$ is also denoted by $+$. For non-negative integer n and a group element $g \in \mathbb{G}$, $n \cdot g$ denotes the n times summation of g , i.e., $\sum_{i=1}^n g$. For a subset $S \in \mathbb{G}$, $\langle s \rangle_{s \in S}$ denotes the subgroup of \mathbb{G} generated by S . We also write $\langle S \rangle$ for short.

2.2 Function Secret Sharing

In this paper, function class \mathcal{F} is a subset of $\mathbb{G}^{\mathcal{X}}$. That is, all functions $f \in \mathcal{F}$ have the same domain \mathcal{X} and the same range \mathbb{G} . Also, since we focus on the information-theoretic setting, it is not required for $f \in \mathcal{F}$ to be computed in polynomial time.

We naturally modify the definition of (2-party) FSS in [2] to the information-theoretic setting as follows:

Definition 1. *A perfectly secure Information-Theoretic 2-party Function Secret Sharing (IT-FSS) scheme $(\text{Gen}, \text{Eval}, \mathcal{K})$ for function class $\mathcal{F} \subseteq \mathbb{G}^{\mathcal{X}}$, consists of a probabilistic algorithm Gen , a deterministic algorithm Eval , and a key space \mathcal{K} with the following syntax:*

- $\text{Gen}(f)$: Taking $f \in \mathcal{F}$ as input, outputs key pair $(k_0, k_1) \in \mathcal{K} \times \mathcal{K}$.
- $\text{Eval}(i, k_i, x)$: Taking party id $i \in \{0, 1\}$, key $k_i \in \mathcal{K}$, and $x \in \mathcal{X}$ as input, outputs share $y_i \in \mathbb{G}$.

satisfying the following correctness and security requirements:

- **Correctness** : For all $f \in \mathcal{F}, x \in \mathcal{X}$,

$$\Pr[y_0 + y_1 = f(x) \mid (k_0, k_1) \leftarrow \text{Gen}(f), y_i \leftarrow \text{Eval}(i, k_i, x)] = 1.$$

- **Security** : For $(k_0, k_1) \leftarrow \text{Gen}(f)$, the distribution of each k_i alone is independent of f .

3 Known Negative Results

In this section, We review the known negative results regarding IT-FSS shown in [2, 9]. Also, we explain that these negative results are not robust, that is, they become not applicable once we slightly modify the function class.

3.1 Lower Bound for Information-Theoretic DPF

Gilboa et al. stated that information-theoretic DPF needs exponential key size [9]. More specifically, let f_α for $\alpha \in \{0, 1\}^n$ be a function $\{0, 1\}^n \rightarrow \{0, 1\}$ such that $f_\alpha(x) = 1$ when $x = \alpha$ and $f_\alpha(x) = 0$ otherwise. Then, the result of [9] implies that an information-theoretic DPF, i.e., an IT-FSS scheme for the function class $\{f_\alpha\}_{\alpha \in \{0, 1\}^n}$ needs $\Omega(2^n)$ bits key size. This lower bound is proved by

showing a connection to a known lower bound of information-theoretic 2-server binary PIR, which is a type of PIR scheme in which the server’s response is just a bit: Similarly to Theorem 2 in [9], we can construct an information-theoretic 2-server binary PIR scheme with $2(\ell + 1)$ bits communication using IT-FSS scheme for the function class $\{f_\alpha\}_{\alpha \in \{0,1\}^n}$ with ℓ bits key size. On the lower bound for information-theoretic 2-server binary PIR, [13] shows that $\Omega(2^n)$ bits communication is needed. Therefore, information-theoretic DPF also needs $\Omega(2^n)$ bits key size.

However, the argument of [9] becomes not immediately applicable when the function class is just slightly modified. For example, on a DPF scheme whose domain and range are $\{0, 1\}^n$, the above negative result does not directly hold since the lower bound for 2-server binary PIR cannot be applied.

3.2 Connection to One-Way Function

Gilboa et al. showed that a computational DPF scheme implies the existence of one-way functions [9]. On the other hand, as a result covering more general function classes, Boyle et al. showed that a computational FSS scheme for a *poly-spanning* function class implies the existence of pseudorandom functions [2]. We explain the latter result.

Let the domain (the range, resp.) of functions in \mathcal{F} be $\{0, 1\}^n$ (\mathbb{G} , resp.). \mathcal{F} is called poly-spanning if \mathcal{F} “efficiently spans” the whole function space. That is, for arbitrary polynomially many input-output pairs $(x_i, y_i) \in \{0, 1\}^n \times \mathbb{G}$, there exist polynomially many functions $f_j \in \mathcal{F}$ such that $y_i = \sum_j f_j(x_i)$ holds for all i . In [2], multi-bit point functions and comparison functions are mentioned as examples of poly-spanning function classes:

- **Multi-bit Point Functions.** The class of functions $\{f_{\alpha,\beta}\}$ over $\alpha \in \{0, 1\}^n$ and $\beta \in \{0, 1\}^m$ where $f_{\alpha,\beta}$ is a function such that $f_{\alpha,\beta}(\alpha) = \beta$ and $f_{\alpha,\beta}(x) = 0$ for $x \neq \alpha$.
- **Comparison Functions.** The class of functions $\{f_\alpha\}$ over $\alpha \in [2^n]$ where $f_\alpha: [2^n] \rightarrow \{0, 1\}$ is a function such that $f_\alpha(x) = 1$ if $x \leq \alpha$ and $f_\alpha(x) = 0$ otherwise.

However, we can make point functions be not poly-spanning by slightly changing the definition. Consider the function class $\{f_{\alpha,1}\}$ over $\alpha \in \{0, 1\}^n$, where the range of each $f_{\alpha,1}$ is \mathbb{Z}_{2^n} , which is a type of point functions used in secure equality test [4] or private queries [10, 12]. This function class is not poly-spanning: In fact, we cannot select polynomially many (with respect to n) functions $f_{\alpha_j,1}$ such that $2^n - 1 = \sum_j f_{\alpha_j,1}(0)$.

4 Our Explicit Lower Bound

In this section, we show an explicit lower bound for IT-FSS. More specifically, we demonstrate the following theorem:

Theorem 1. *Let $(\text{Gen}, \text{Eval}, \mathcal{K})$ be an IT-FSS scheme for a function class \mathcal{F} . Then, the key size is bounded below as following:*

$$\log |\mathcal{K}| \geq -1 + \log |\langle f - f' \rangle_{f, f' \in \mathcal{F}}|.$$

If \mathbb{G} is an elementary Abelian 2-group (i.e., $g + g = 0$ for all $g \in \mathbb{G}$), then the key size is more strictly bounded below as following:

$$\log |\mathcal{K}| \geq -1 + \log |\langle \mathcal{F} \rangle|.$$

We give this explicit lower bound for IT-FSS in two steps:

1. We convert an IT-FSS scheme to a *simplified* IT-FSS scheme (defined later) with (at most) 1-bit additional key size (Theorem 2).
2. We give an explicit lower bound for a simplified IT-FSS scheme (Corollary 1 and Corollary 2).

Theorem 1 is readily deduced from Theorem 2, Corollary 1, and Corollary 2. We explain the first step in Sect. 4.1, and the second step in Sect. 4.2. Finally, we give examples for some specific function classes in Sect. 4.3.

4.1 Conversion from IT-FSS to Simplified IT-FSS

We define *simplified* IT-FSS as follows.

Definition 2. *A simplified IT-FSS scheme is an IT-FSS scheme $(\text{Gen}, \text{Eval}, \mathcal{K})$ with the following four properties:*

1. *Eval algorithm does not depend on party id, i.e., $\text{Eval}(0, k, x) = \text{Eval}(1, k, x)$ for all $k \in \mathcal{K}$ and $x \in \mathcal{X}$ (we write them simply as $\text{Eval}(k, x)$).*
2. *The possible key pairs are symmetric, i.e., if $\Pr[(k, k') \leftarrow \text{Gen}(f)] > 0$, then $\Pr[(k', k) \leftarrow \text{Gen}(f)] > 0$.*
3. *Different keys define different evaluation functions, i.e., for all $k \neq k' \in \mathcal{K}$, there exists $x \in \mathcal{X}$ such that $\text{Eval}(k, x) \neq \text{Eval}(k', x)$.*
4. *The key space is not redundant, i.e., for all $k \in \mathcal{K}$ and $f \in \mathcal{F}$, there exists $k' \in \mathcal{K}$ such that $\Pr[(k, k') \leftarrow \text{Gen}(f)] > 0$.*

Given an arbitrary IT-FSS scheme $(\text{Gen}, \text{Eval}, \mathcal{K})$ for function class \mathcal{F} , we construct a simplified IT-FSS scheme $(\text{Gen}', \text{Eval}', \mathcal{K}')$ such that $\log |\mathcal{K}'| \leq 1 + \log |\mathcal{K}|$. First, we construct an IT-FSS scheme with the first and the second properties of Definition 2. Secondly, we construct an IT-FSS scheme with the first, the second, and the third properties. Finally, we construct a simplified IT-FSS scheme.

First Step: Conversion to IT-FSS with Properties 1 and 2. We construct an IT-FSS scheme $(\text{Gen}_1, \text{Eval}_1, \mathcal{K}_1)$ with the first and the second properties of Definition 2 from the original IT-FSS scheme $(\text{Gen}, \text{Eval}, \mathcal{K})$. These properties can be easily satisfied by putting party ids into keys as follows: Let \mathcal{K}_1 be $\{0, 1\} \times \mathcal{K}$.

$\text{Gen}_1(f)$ generates (k_0, k_1) by invoking $\text{Gen}(f)$, randomly chooses $b \in \{0, 1\}$, and outputs $(k'_0, k'_1) = (b||k_b, (b \oplus 1)||k_{b \oplus 1})$. $\text{Eval}_1(k', x)$ parses the key k' into $b||k$, and outputs the value $y = \text{Eval}(b, k, x)$. The requirements of correctness and security for $(\text{Gen}_1, \text{Eval}_1, \mathcal{K}_1)$ are readily deduced from those for $(\text{Gen}, \text{Eval}, \mathcal{K})$. The first property of Definition 2 is satisfied since Eval_1 does not take a party id as input. The second property of Definition 2 is satisfied since a key pair generated by $\text{Gen}(f)$ is flipped when the (random) party id $b \in \{0, 1\}$ generated in $\text{Gen}_1(f)$ is flipped. On the key size, $\log |\mathcal{K}_1|$ is equal to $1 + \log |\mathcal{K}|$.

Second Step: Conversion to IT-FSS with Properties 1, 2, and 3. Then, we convert the IT-FSS scheme $(\text{Gen}_1, \text{Eval}_1, \mathcal{K}_1)$ to $(\text{Gen}_2, \text{Eval}_2, \mathcal{K}_2)$ with the first, the second, and the third properties of Definition 2. We define an equivalence relation \sim on \mathcal{K}_1 as follows:

$$k \sim k' \Leftrightarrow \text{Eval}_1(k, x) = \text{Eval}_1(k', x) \text{ for all } x \in \mathcal{X}.$$

Let $\mathcal{K}_2 \subseteq \mathcal{K}_1$ be a complete system of representatives, and let ϕ be the natural surjection $\mathcal{K}_1 \rightarrow \mathcal{K}_2$. That is, $\phi(k) \sim k$ holds for all $k \in \mathcal{K}_1$. $\text{Gen}_2(f)$ generates (k_0, k_1) by invoking $\text{Gen}_1(f)$ and outputs $(\phi(k_0), \phi(k_1))$. Eval_2 is the same as Eval_1 . The requirements of correctness and security for $(\text{Gen}_2, \text{Eval}_2, \mathcal{K}_2)$ are readily deduced from those for $(\text{Gen}_1, \text{Eval}_1, \mathcal{K}_1)$. Also, the first and the second properties of Definition 2 are readily deduced from those for $(\text{Gen}_1, \text{Eval}_1, \mathcal{K}_1)$. The third property of Definition 2 is satisfied by the definition of \mathcal{K}_2 . On the key size, $\log |\mathcal{K}_2|$ is at most $\log |\mathcal{K}_1|$.

Third Step: Conversion to Simplified IT-FSS. Finally, we convert the IT-FSS scheme $(\text{Gen}_2, \text{Eval}_2, \mathcal{K}_2)$ to a simplified IT-FSS scheme $(\text{Gen}', \text{Eval}', \mathcal{K}')$. Let $\mathcal{K}_{2,b,f} \subseteq \mathcal{K}_2$ for $b \in \{0, 1\}$ and $f \in \mathcal{F}$ be the set of keys that may appear as party b 's key corresponding to f . That is,

$$\begin{aligned} \mathcal{K}_{2,0,f} &= \{k \in \mathcal{K}_2 \mid \exists k' \in \mathcal{K}_2 \text{ s.t. } \Pr[(k, k') \leftarrow \text{Gen}_2(f)] > 0\}, \\ \mathcal{K}_{2,1,f} &= \{k \in \mathcal{K}_2 \mid \exists k' \in \mathcal{K}_2 \text{ s.t. } \Pr[(k', k) \leftarrow \text{Gen}_2(f)] > 0\}. \end{aligned}$$

From the security requirement for $(\text{Gen}_2, \text{Eval}_2, \mathcal{K}_2)$, the distribution of party b 's key corresponding to f is independent of f . Therefore, $\mathcal{K}_{2,b,f}$ is also independent of f . $\mathcal{K}_{2,b,f}$ is also independent of b because

$$\Pr[(k, k') \leftarrow \text{Gen}_2(f)] > 0 \Leftrightarrow \Pr[(k', k) \leftarrow \text{Gen}_2(f)] > 0$$

holds from the second property of Definition 2 for $(\text{Gen}_2, \text{Eval}_2, \mathcal{K}_2)$. Let \mathcal{K}' denote $\mathcal{K}_{2,b,f}$. In Gen_2 algorithm, keys in $\mathcal{K}_2 \setminus \mathcal{K}'$ are never generated. Therefore, we can use \mathcal{K}' as a key space instead of \mathcal{K}_2 . The first, the second, and the third properties of Definition 2, and the requirements of correctness and security for $(\text{Gen}_2, \text{Eval}_2, \mathcal{K}')$ are readily deduced from those for $(\text{Gen}_2, \text{Eval}_2, \mathcal{K}_2)$. Also, the fourth property of Definition 2 is satisfied by the definition of \mathcal{K}' . On the key size, $\log |\mathcal{K}'|$ is at most $\log |\mathcal{K}_2|$.

In summary, the following theorem holds:

Theorem 2. *Given an arbitrary IT-FSS scheme $(\text{Gen}, \text{Eval}, \mathcal{K})$ for a function class \mathcal{F} , we can construct a simplified IT-FSS scheme $(\text{Gen}', \text{Eval}', \mathcal{K}')$ for \mathcal{F} such that $\log |\mathcal{K}'| \leq 1 + \log |\mathcal{K}|$.*

4.2 Lower Bound for Simplified IT-FSS

In this section, we give an explicit lower bound for a simplified IT-FSS scheme $(\text{Gen}, \text{Eval}, \mathcal{K})$ for a function class \mathcal{F} in two steps: First, we describe behavior of Gen algorithm using permutations on \mathcal{K} . Then, we construct an injection from $\langle f - f' \rangle_{f, f' \in \mathcal{F}}$ to \mathcal{K} . This means that $|\mathcal{K}|$ is bounded below by $|\langle f - f' \rangle_{f, f' \in \mathcal{F}}|$.

First Step: Description of Gen Algorithm Using Permutations. In the following, we sometimes identify the key space \mathcal{K} with a set $[\mathcal{K}]$. Let P_f be a $|\mathcal{K}| \times |\mathcal{K}|$ matrix such that

$$P_f[k, k'] := \begin{cases} 1 & (\Pr[(k, k') \leftarrow \text{Gen}(f)] > 0) \\ 0 & (\text{otherwise}). \end{cases}$$

Then, the following lemma holds:

Lemma 1. *P_f is a permutation matrix. That is, the number of 1's in each row/column is equal to one.*

Proof. Assume there exist $k, k', k'' \in \mathcal{K}$ such that $k' \neq k''$ and $P_f[k, k'] = P_f[k, k''] = 1$. From the correctness requirement, the followings hold for all $x \in \mathcal{X}$:

$$\begin{aligned} \text{Eval}(k, x) + \text{Eval}(k', x) &= f(x), \\ \text{Eval}(k, x) + \text{Eval}(k'', x) &= f(x). \end{aligned}$$

These equations imply that $\text{Eval}(k', x)$ is equal to $\text{Eval}(k'', x)$ for all $x \in \mathcal{X}$, but this contradicts the third property of Definition 2. Hence, the number of 1's in each row is at most one. Similarly, the number of 1's in each column is at most one. Also, the fourth property and the second property of Definition 2 means the number of 1's in each row/column is at least one. Therefore, the number of 1's in each row/column is equal to one. □

From Lemma 1, there exists a unique permutation σ_f on \mathcal{K} corresponding to permutation matrix P_f , i.e., $P_f[k, \sigma_f(k)] = 1$. Then, the behavior of Gen algorithm can be described using σ_f : $\text{Gen}(f)$ chooses $k \in \mathcal{K}$ according to some distribution, which is independent of f from the security requirement, and outputs $(k, \sigma_f(k))$.

Second Step: Construction of an Injection $\langle f - f' \rangle_{f, f' \in \mathcal{F}} \rightarrow \mathcal{K}$. For $k \in \mathcal{K}$, let $v_k \in \mathbb{G}^{\mathcal{X}}$ be a function mapping $x \in \mathcal{X}$ to $\text{Eval}(k, x) \in \mathbb{G}$. From the correctness requirement, $v_k + v_{\sigma_f(k)}$ is equal to f for all $k \in \mathcal{K}$. Hence, we have

$$\begin{aligned} v_k + v_{\sigma_f(k)} &= f, \\ v_{\sigma_f(k)} + v_{\sigma_{f'} \circ \sigma_f(k)} &= f', \end{aligned}$$

therefore

$$v_k - v_{\sigma_{f'} \circ \sigma_f(k)} = f - f'. \tag{1}$$

Let $\pi_{f, f'}$ be a permutation equal to $\sigma_{f'} \circ \sigma_f$. Since the Eq. (1) holds for all $k \in \mathcal{K}$ and $f, f' \in \mathcal{F}$, we have

$$\begin{aligned} v_k - v_{\pi_{f_1, f'_1}(k)} &= f_1 - f'_1, \\ v_{\pi_{f_1, f'_1}(k)} - v_{\pi_{f_2, f'_2} \circ \pi_{f_1, f'_1}(k)} &= f_2 - f'_2, \\ &\vdots \\ v_{\pi_{f_{n-1}, f'_{n-1}} \circ \dots \circ \pi_{f_1, f'_1}(k)} - v_{\pi_{f_n, f'_n} \circ \dots \circ \pi_{f_1, f'_1}(k)} &= f_n - f'_n, \end{aligned}$$

therefore

$$v_k - v_{\pi_{f_n, f'_n} \circ \dots \circ \pi_{f_1, f'_1}(k)} = \sum_{i=1}^n (f_i - f'_i) \tag{2}$$

for all $n \in \mathbb{N}$, $k \in \mathcal{K}$, and $f_1, \dots, f_n, f'_1, \dots, f'_n \in \mathcal{F}$. Let $\Phi_k : \langle f - f' \rangle_{f, f' \in \mathcal{F}} \rightarrow \mathcal{K}$ be a function defined as

$$\Phi_k \left(\sum_{i=1}^n (f_i - f'_i) \right) = \pi_{f_n, f'_n} \circ \dots \circ \pi_{f_1, f'_1}(k).$$

We note that some of the elements f_1, \dots, f_n in the definition of Φ_k may be duplicate, and similarly for f'_1, \dots, f'_n . Then, the following lemma holds:

Lemma 2. *For all $k \in \mathcal{K}$, Φ_k is well-defined and injective.*

Proof. First, we prove the well-definedness. Let $\sum_{i=1}^n (f_i - f'_i)$ and $\sum_{j=1}^m (g_j - g'_j)$, where $f_i, f'_i, g_j, g'_j \in \mathcal{F}$ for all i and j , be two expressions for the same element of $\langle f - f' \rangle_{f, f' \in \mathcal{F}}$ and let k_f and k_g be $\pi_{f_n, f'_n} \circ \dots \circ \pi_{f_1, f'_1}(k)$ and $\pi_{g_m, g'_m} \circ \dots \circ \pi_{g_1, g'_1}(k)$ respectively. From the Eq. (2), we have

$$\begin{aligned} v_{k_f} &= v_k - \sum_{i=1}^n (f_i - f'_i) \\ &= v_k - \sum_{j=1}^m (g_j - g'_j) \\ &= v_{k_g}. \end{aligned}$$

This means that $\text{Eval}(k_f, x)$ is equal to $\text{Eval}(k_g, x)$ for all $x \in \mathcal{X}$. From the third property of Definition 2, k_f is equal to k_g and Φ_k is well-defined.

Since Φ_k is well-defined, we have $v_k - v_{\Phi_k(h)} = h$ for all $h \in \langle f - f' \rangle_{f, f' \in \mathcal{F}}$ from the Eq. (2). Therefore, $h \in \langle f - f' \rangle_{f, f' \in \mathcal{F}}$ is uniquely determined by $\Phi_k(h)$ and this means that Φ_k is injective. \square

This lemma gives an explicit lower bound for a simplified IT-FSS scheme:

Corollary 1. *Let $(\text{Gen}, \text{Eval}, \mathcal{K})$ be a simplified IT-FSS scheme for a function class \mathcal{F} . Then, the size of key space \mathcal{K} is bounded below as following:*

$$|\mathcal{K}| \geq |\langle f - f' \rangle_{f, f' \in \mathcal{F}}|.$$

Proof. This corollary is readily deduced from the fact that by Lemma 2 there exists an injection $\Phi : \langle f - f' \rangle_{f, f' \in \mathcal{F}} \rightarrow \mathcal{K}$. □

For the Case of Elementary Abelian 2-Groups. When \mathbb{G} is an elementary Abelian 2-group, i.e., $g + g = 0$ for all $g \in \mathbb{G}$, we can give a more strict lower bound than Corollary 1. From the correctness and the fact that $-g = g$ in \mathbb{G} , we have

$$v_k - v_{\sigma_f(k)} = f$$

for all $k \in \mathcal{K}$ and all $f \in \mathcal{F}$. Now an argument similar to the derivation of Eq. (2) from Eq. (1) implies that we have

$$v_k - v_{\sigma_{f_n} \circ \dots \circ \sigma_{f_1}(k)} = \sum_{i=1}^n f_i.$$

Let $\Psi_k : \langle \mathcal{F} \rangle \rightarrow \mathcal{K}$ be a function defined as

$$\Psi_k \left(\sum_{i=1}^n f_i \right) = \sigma_{f_n} \circ \dots \circ \sigma_{f_1}(k).$$

Similarly to Lemma 2, Ψ_k is also well-defined and injective. Therefore, we have a more strict lower bound:

Corollary 2. *Let $(\text{Gen}, \text{Eval}, \mathcal{K})$ be a simplified IT-FSS scheme for a function class \mathcal{F} . If \mathbb{G} is an elementary Abelian 2-group, then the size of key space \mathcal{K} is bounded below as following:*

$$|\mathcal{K}| \geq |\langle \mathcal{F} \rangle|.$$

4.3 Examples for Specific Function Classes

We show several examples for specific function classes. The following function classes are slightly modified from the original ones and are not poly-spanning. Therefore the negative result [2] reviewed in Sect. 3.2 cannot be applied to these function classes. Note that these function classes are also used in applications such as secure equality test and secure integer comparison [4].

Point Functions [2, 3, 9]. The function class is defined as follows:

- $\mathcal{X} = \mathbb{G} = \mathbb{Z}_{2^n}$.

– $\mathcal{F} = \{f_\alpha\}_{\alpha \in \mathbb{Z}_{2^n}}$, where $f_\alpha(x) = 1$ if $x = \alpha$ and $f(x) = 0$ otherwise.

For this function class \mathcal{F} , we can determine $\langle f - f' \rangle_{f, f' \in \mathcal{F}}$ as following:

$$\langle f - f' \rangle_{f, f' \in \mathcal{F}} = \left\{ f \in \mathbb{G}^{\mathcal{X}} \mid \sum_{x \in \mathcal{X}} f(x) = 0 \right\}.$$

Therefore, the key size of an IT-FSS scheme for this \mathcal{F} is bounded below by $-1 + n(2^n - 1)$ bits.

Distributed Comparison Functions [1–3]. The function class is defined as follows:

- $\mathcal{X} = \mathbb{G} = \mathbb{Z}_{2^n}$.
- $\mathcal{F} = \{f_\alpha\}_{\alpha \in \mathbb{Z}_{2^n}}$, where $f_\alpha(x) = 1$ if $x \leq \alpha$ and $f(x) = 0$ otherwise.

For this function class \mathcal{F} , we can determine $\langle f - f' \rangle_{f, f' \in \mathcal{F}}$ as following:

$$\langle f - f' \rangle_{f, f' \in \mathcal{F}} = \{f \in \mathbb{G}^{\mathcal{X}} \mid f(0) = 0\}.$$

Indeed, for any $1 \leq \alpha \leq 2^n - 1$, $g := f_\alpha - f_{\alpha-1}$ satisfies that $g(\alpha) = 1$ and $g(x) = 0$ for $x \neq \alpha$; therefore the elements of $\langle f - f' \rangle_{f, f' \in \mathcal{F}}$ have no restrictions on the values at $x \neq 0$. On the other hand, for any $f, f' \in \mathcal{F}$ we always have $f(0) - f'(0) = 1 - 1 = 0$. Therefore, the key size of an IT-FSS scheme for this \mathcal{F} is bounded below by $-1 + n(2^n - 1)$ bits.

Bit-Conjunction Functions [1]. The function class is defined as follows:

- $\mathcal{X} = \{0, 1\}^n, \mathbb{G} = \mathbb{Z}_{2^n}$.
- $\mathcal{F} = \{f_S\}_{S \subseteq \{1, 2, \dots, n\}}$, where $f_S(x) = 1$ if $\wedge_{i \in S} x_i = 1$ and $f(x) = 0$ otherwise.

For this function class \mathcal{F} , we can determine $\langle f - f' \rangle_{f, f' \in \mathcal{F}}$ as following:

$$\langle f - f' \rangle_{f, f' \in \mathcal{F}} = \{f \in \mathbb{G}^{\mathcal{X}} \mid f(1, 1, \dots, 1) = 0\}.$$

Indeed, for any $0 \leq k \leq n - 1$ and any $x = (x_1, \dots, x_n) \in \{0, 1\}^n$ with Hamming weight k , let $S = \{i \mid x_i = 1\}$. Then $g := f_S - f_{\{1, 2, \dots, n\}}$ satisfies that $g(x) = 1$, $g(1, 1, \dots, 1) = 1 - 1 = 0$, and $g(y) = 0$ for any $y \in \{0, 1\}^n \setminus \{x\}$ with Hamming weight at most k . Therefore, for functions in $\langle f - f' \rangle_{f, f' \in \mathcal{F}}$, the value at x can be freely adjusted without changing the values at points in $\{y \mid \text{Hamming weight of } y \text{ is } n \text{ or at most } k\} \setminus \{x\}$. Iterating this process for $k = 0, \dots, n - 1$, the values at all $x \neq (1, 1, \dots, 1)$ can be freely adjusted. On the other hand, for any $f, f' \in \mathcal{F}$ we always have $f(1, 1, \dots, 1) - f'(1, 1, \dots, 1) = 1 - 1 = 0$.

Therefore, the key size of an IT-FSS scheme for this \mathcal{F} is bounded below by $-1 + n(2^n - 1)$ bits.

5 Nearly Optimal Construction

We prove the following theorem by concretely constructing a scheme as in the statement.

Theorem 3. *Let \mathcal{F} be an arbitrary function class. Then there exists an IT-FSS scheme $(\text{Gen}, \text{Eval}, \mathcal{K})$ for the function class \mathcal{F} such that*

$$\log |\mathcal{K}| = \log |\langle \mathcal{F} \rangle|.$$

Proof. We construct an IT-FSS scheme $(\text{Gen}, \text{Eval}, \mathcal{K})$ for a wider function class $\langle \mathcal{F} \rangle$, instead of \mathcal{F} . Since $\langle \mathcal{F} \rangle$ is closed under addition, an additive secret sharing for $\langle \mathcal{F} \rangle$ can be regarded as an IT-FSS scheme for $\langle \mathcal{F} \rangle$. That is, $\text{Gen}(f)$ randomly chooses $(k_0, k_1) \in \langle \mathcal{F} \rangle \times \langle \mathcal{F} \rangle$ such that $k_0 + k_1 = f$. $\text{Eval}(b, k, x)$ outputs $k(x)$. In this scheme, the key space \mathcal{K} is $\langle \mathcal{F} \rangle$ and the key size is $\log |\langle \mathcal{F} \rangle|$. \square

For the key size in the statement in comparison to our lower bounds, let f_0 be an element of \mathcal{F} with minimal order (in the Abelian group $\mathbb{G}^{\mathcal{X}}$) and let η denote the order of f_0 . Let \mathcal{F}' be the subgroup of $\langle \mathcal{F} \rangle$ generated by $\{f_0\} \cup \{f - f'\}_{f, f' \in \mathcal{F}}$. Then we have $g = f_0 + (g - f_0) \in \mathcal{F}'$ for all $g \in \mathcal{F}$, therefore $\mathcal{F}' = \langle \mathcal{F} \rangle$. Moreover, since f_0 has order η , for all $g \in \mathcal{F}'$, there exists $a \in [\eta]$ such that $g \in a \cdot f_0 + \langle f - f' \rangle_{f, f' \in \mathcal{F}}$. This implies that $|\langle \mathcal{F} \rangle| \leq \eta \cdot |\langle f - f' \rangle_{f, f' \in \mathcal{F}}|$, therefore the difference of the key size in our construction and our lower bound in Theorem 1 is

$$\log |\langle \mathcal{F} \rangle| - (-1 + \log |\langle f - f' \rangle_{f, f' \in \mathcal{F}}|) \leq 1 + \log \eta \text{ (bits)}$$

for the general case, and it is only

$$\log |\langle \mathcal{F} \rangle| - (-1 + \log |\langle \mathcal{F} \rangle|) \leq 1 \text{ (bit)}$$

for the case of elementary Abelian 2-group \mathbb{G} . The right-hand side $1 + \log \eta$ is at most $1 + \log e_{\mathbb{G}}$ where $e_{\mathbb{G}}$ denotes the exponent of \mathbb{G} , that is, the minimal positive integer $e_{\mathbb{G}}$ with $e_{\mathbb{G}} \cdot g = 0$ for all $g \in \mathbb{G}$ (note that $e_{\mathbb{G}} \cdot f_0 = 0$ and therefore $\eta \leq e_{\mathbb{G}}$). In particular, since $e_{\mathbb{G}} \leq |\mathbb{G}|$, the difference of the key size in our construction and our lower bound (for the general case) is at most $1 + \log |\mathbb{G}|$ bits. For function classes given in Sect. 4.3, this $1 + \log |\mathbb{G}|$ bits gap is just $n + 1$ bits and small enough in comparison to the lower bound ($\simeq n2^n$ bits).


Acknowledgements. This research was in part conducted under a contract of “Research and development on new generation cryptography for secure wireless communication services” among “Research and Development for Expansion of Radio Wave Resources (JPJ000254),” which was supported by the Ministry of Internal Affairs and Communications, Japan. This research was also partially supported by JSPS KAKENHI Grant Numbers JP19H01109, JP21J20186 and JP22K11906, JST CREST Grant Number JPMJCR2113, and JST AIP Acceleration Research JPMJCR22U5.

References

1. Boyle, E., et al.: Function secret sharing for mixed-mode and fixed-point secure computation. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 871–900. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_30
2. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_12
3. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS2016, pp. 1292–1303. ACM (2016). <https://doi.org/10.1145/2976749.2978429>
4. Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 341–371. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_14
5. Bunn, P., Katz, J., Kushilevitz, E., Ostrovsky, R.: Efficient 3-party distributed ORAM. In: Galdi, C., Kolesnikov, V. (eds.) SCN 2020. LNCS, vol. 12238, pp. 215–232. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57990-6_11
6. Bunn, P., Kushilevitz, E., Ostrovsky, R.: CNF-FSS and its applications. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography – PKC 2022. PKC 2022. LNCS, vol. 13177, pp. 283–314. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97121-2_11
7. Doerner, J., Shelat, A.: Scaling oram for secure computation. In: CCS2017, pp. 523–535. ACM (2017). <https://doi.org/10.1145/3133956.3133967>
8. Eskandarian, S., Corrigan-Gibbs, H., Zaharia, M., Boneh, D.: Express: lowering the cost of metadata-hiding communication with cryptographic privacy. In: 30th USENIX, pp. 1775–1792. USENIX Association (2021)
9. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_35
10. Hayata, J., Schuldt, J.C.N., Hanaoka, G., Matsuura, K.: On private information retrieval supporting range queries. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12309, pp. 674–694. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59013-0_33
11. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 600–620. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_34
12. Wang, F., Yun, C., Goldwasser, S., Vaikuntanathan, V., Zaharia, M.: Splinter: Practical private queries on public data. In: NSDI2017, pp. 299–313. USENIX Association (2017)
13. Wehner, S., de Wolf, R.: Improved lower bounds for locally decodable codes and private information retrieval. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1424–1436. Springer, Heidelberg (2005). https://doi.org/10.1007/11523468_115



Multi-Theorem Fiat-Shamir Transform from Correlation-Intractable Hash Functions

Michele Ciampi^(✉)  and Yu Xia

The University of Edinburgh, Edinburgh, UK
`{michele.ciampi,yu.xia}@ed.ac.uk`

Abstract. In STOC 2019 Canetti et al. showed how to soundly instantiate the Fiat-Shamir transform assuming that prover and verifier have access to the key of a *correlation intractable hash function for efficiently searchable relations*. The transform requires the starting protocol to be a special 3-round public-coin scheme that Canetti et al. call *trapdoor sigma-protocol*. One downside of the Canetti et al. approach is that the key of the hash function can be used only once (or a pre-determined bounded number of times). That is, each new zero-knowledge proof requires a freshly generated hash key (i.e., a freshly generated setup). This is in contrast to what happens with the standard Fiat-Shamir transform, where the prover, having access to the same hash function (modelled as a random-oracle), can generate an unbounded number of proofs that are guaranteed to be zero-knowledge and sound.

As our main contribution we extend the results of Canetti et al., by proposing a multi-theorem protocol that follows the Fiat-Shamir paradigm and relies on correlation intractable hash functions. Moreover, our protocol remains zero-knowledge and sound even against adversaries that choose the statement to be proven (and the witness for the case of zero-knowledge) adaptively on the key of the hash function. Our construction is presented in the form of a compiler, that follows the Fiat-Shamir paradigm, which takes as input any trapdoor sigma-protocol for the NP-language L and turns it into a non-interactive zero-knowledge protocol that satisfies the properties we mentioned. To be best of our knowledge, ours is the first compiler that follows the Fiat-Shamir paradigm to obtain a multi-theorem adaptive NIZK relying on correlation intractable hash functions.

Keywords: NIZK · Fiat-Shamir Transform · Adaptive Multi-Theorem Zero-Knowledge · Correlation Intractable Hash Functions

1 Introduction

Non-interactive zero-knowledge (NIZK) proofs [2, 13] allow a prover to convince a verifier about the validity of an NP-statement with just one round of interaction (one message that goes from the prover to the verifier). One of the most famous techniques used to realize non-interactive proofs is the *Fiat-Shamir (FS) transform* [16]. This transform takes as input a *sigma-protocol* and turns it into

a NIZK proof. A sigma-protocol is a special three-round public-coin interactive proof executed between a prover P and a verifier V , where P 's goal is to convince V that a common statement x belongs to a given NP language L . The prover knows a witness w (corresponding to x) and starts the interaction by sending a first message a ; the verifier then sends a uniformly random bit-string c , called the challenge, to which the prover replies with the last message z . Finally, the verifier decides whether $x \in L$ or not based on x and the transcript (a, c, z) .

The FS transform makes a sigma-protocol non-interactive by letting the prover do the sampling of the challenge. In particular, the prover computes $c \leftarrow H(a)$, where H is a hash function. One way to argue about the security of this construction is by modeling H as a Random Oracle [1, 14]. Recently, [3–6, 18, 19, 22] showed that if the hash function is correlation-intractable (CI) for certain relations, then the resulting NIZK is sound. Informally, the CI property ensures that given a random hash key k , it is computationally difficult to find any input α , s.t. $(\alpha, H_k(\alpha)) \in R$ for a particular relation R .

In more detail, Canetti et al. [4] shows that the FS transform remains secure assuming that the hash function is correlation intractable for *efficiently searchable relations*¹. The result of [4] can be applied only to a restricted class of sigma-protocols called *trapdoor sigma-protocol*. Trapdoor sigma-protocols are three-round public-coin protocols defined in the Common Reference String (CRS) model that enjoy three main properties: honest verifier zero-knowledge (HVZK), optimal soundness, and admit a *bad-challenge* extractor. The property of HVZK is quite standard and guarantees the existence of a simulator that, upon receiving the challenge (the second round), it produces a transcript that is indistinguishable from the transcript generated via the interaction of an honest prover and verifier. Optimal soundness guarantees that for any statement $x \notin L$ and the first-round message a there exists at most one challenge c , such that a verifier would accept the transcript (a, c, z) , for the statement x , for some third-round z . We refer to the unique challenge c as the *bad-challenge*. Finally, the bad-challenge extractor is an algorithm that takes as input a false statement x , a valid first-round a , and some trapdoor information τ , and efficiently computes the bad-challenge c .

Adaptive multi-theorem NIZK. The most basic notion of soundness for a non-interactive proof system guarantees soundness in the presence of an adversary that decides the statement to be proven before the sampling of the CRS. Similarly, the notion of zero-knowledge is guaranteed to hold for any choice of theorem-witness sampled by the adversary non-adaptively on the CRS. We refer to this class of adversaries as *non-adaptive adversaries*. It is possible to consider stronger (and more realistic) notions of security that guarantee that both the soundness and the zero-knowledge hold even if the adversary can make the choice of the theorem to be proven (and of the witness for the zero-knowledge experiment) adaptively on the CRS. In [4] the authors argue that if the trapdoor

¹ A relation is efficiently searchable if given x it is efficient to find y such that $(x, y) \in R$.

sigma-protocol admits a special bad-challenge extractor, and moreover it is *adaptive* special-honest verifier zero-knowledge², then the NIZK they obtain using CI hash functions is also adaptive secure. Unfortunately, the only trapdoor sigma-protocol known to satisfy all the required properties is the Lapidot-Shamir [20] protocol for Hamiltonian graphs. In [9] the authors show that all sigma-protocols can be turned into trapdoor sigma-protocols with an adaptive HVZK simulator. One drawback of all the previous approaches is that the zero-knowledge property is not preserved if the same hash key is used to generate more than one proof. However, we would like to be able to use the same hash key to generate multiple proofs (for potentially different theorems). We refer to this notion of zero-knowledge as *multi-theorem NIZK*, and we investigate the following question:

Is it possible to obtain an adaptive multi-theorem NIZK by applying the Fiat-Shamir paradigm using a hash function that is correlation intractable for efficiently searchable relations?

Another way to phrase the above is that we ask whether it is possible to construct an adaptive multi-theorem NIZK using the same setup (and complexity) assumption as in [4, 9].

1.1 Our Results

In this work we show how to obtain an *adaptive multi-theorem* NIZK for any language L that admits a trapdoor sigma-protocol Σ_L (we do not require Σ_L to be adaptive HVZK). The nice feature of our NIZK is that the prover, after a pre-processing (non-interactive) phase, upon receiving the statement to be proven and the corresponding witness, generates proofs by just following the FS paradigm.

Due to its FS-like structure, the soundness of our scheme relies only on the security of the underlying trapdoor sigma-protocols and on the correlation-intractability of the hash function (exactly as in all previous works that although achieved a weaker form of zero-knowledge). The zero-knowledge property instead relies on the HVZK of the trapdoor sigma-protocols, the security of the CI hash function, and the hardness of the Decisional Diffie-Hellman (DDH) assumption. This is exactly in the same spirit as [4, 9] where the authors instead rely on the hardness of public-key encryption schemes to argue about zero-knowledge. An informal theorem that summarizes our result is the following

Theorem (informal): *If Σ_L is a trapdoor sigma-protocol for the language L , then it is possible to realize an adaptive multi-theorem NIZK protocol that follows the FS paradigm. In particular, the soundness of the NIZK protocol depends only on the soundness of underlying trapdoor sigma-protocols and on the security of the hash function.*

² The notion of adaptive HVZK guarantees the existence of a simulator that can generate the first-round of the protocol without the knowledge of the theorem.

We note that an easy way to construct a multi-theorem NIZK would be to use the OR approach proposed in [15]. In this, a statement $T \notin L^*$ for a membership-hard language³ L^* is put in the CRS, and the prover provides an OR proof proving that either $x \in L$ or $T \in L^*$. This approach has two main drawbacks: 1) the NIZK is inherently computational zero-knowledge and 2) the soundness holds only under the condition that the tuple T is sampled such that $T \notin L^*$. In our work, we show how to modify the FLS approach to remove the second limitation. Hence, we obtain a NIZK that has exactly the same setup assumptions as previous works, but in addition, we obtain a protocol that is multi-theorem.

1.2 Technical Overview

Adaptive Multi-theorem NIZK from CI Hash Functions. We first recall the approach proposed in [15] used to realize an adaptive multi-theorem NIZK protocol for an NP language L . In this, the prover generates an OR proof showing that either $x \in L$ or that $T \in L^*$, where T is an instance that is part of the CRS. The soundness holds due to the soundness of the OR proof and the fact that by the construction of the CRS $T \notin L^*$. The adaptive zero-knowledge comes from the fact that a simulator, to generate simulated proofs needs to program the CRS with $T^* \in L^*$ (and for this no knowledge about the statement to be proven is needed). Upon receiving a statement x , the simulator uses the witness for T^* to generate the OR proof. If the OR proof is witness-indistinguishable (WI), and L^* is a membership-hard language, then the protocol is adaptive zero-knowledge. The multi-theorem feature comes from the fact that the WI property is closed under sequential composition.

By relying on the result of [11], it is possible to compile two sigma-protocols, respectively for the language L_1 and L_2 , into a new sigma-protocol for the OR language $L_1 \vee L_2$. In this paper, we argue that the compiler of [11] works similarly for trapdoor sigma-protocols. This means that if we have a trapdoor sigma-protocol for L and one for L^* , we can obtain an adaptive multi-theorem NIZK protocol by doing the following. First, we obtain a trapdoor sigma-protocol for the language $L \vee L^*$, and then we apply the FS transform to the resulting protocol thus obtaining a NIZK protocol for the language $L \vee L^*$.

The scheme we have just described departs from the FS paradigm mostly due to the presence of the T value embedded in the CRS (that the simulator needs to program as we have discussed earlier). In the FS paradigms, such a component is not required, since the simulator only needs to program the hash function to perform the final simulation. But more importantly, the value T needs to be correctly generated, (i.e., it must not belong to L^* otherwise the soundness does not hold). This is clearly something undesirable since now the soundness does not only rely on the security of the hash function (which is the case for the FS transform) but also requires additional parameters to be generated honestly.

³ Intuitively, a membership-hard language is one for which it is possible to sample instances of the problem in a way that it is hard to detect if a given instance is in the language or not.

We work around this problem as follows. We define L^* as being the language of all the DH tuples, and instead of requiring the CRS to contain $T \notin L^*$, we let the prover pick the tuple T . We then require the prover to provide a non-interactive zero-knowledge proof via a protocol Π_{NDH} thus proving that the tuple does not belong to L^* (i.e., T is non-DH). Note that we require Π_{NDH} to be a NIZK protocol that guarantees security only if one proof is generated (i.e., it is not multi-theorem zero-knowledge). In particular, Π_{NDH} can be instantiated via the Fiat-Shamir transform using a correlation intractable hash function on a specific trapdoor sigma-protocol (we will elaborate more on this in the technical part of the paper). The rest of the protocol follows as before. That is, the prover, upon receiving a statement x and its witness, perform an OR proof, proving either that $x \in L$ or that T is a DH tuple.

The main observation here is that Π_{NDH} needs to be run only once, and the obtained proof can be reused any time the prover is required to generate a proof for a new instance x . So, we can see our protocol as divided into two phases. In the *offline phase* the prover samples a non-DH tuple T , and runs Π_{NDH} to generate a NIZK proof that we denote with π^{NDH} (without sending it). Upon receiving a statement and a witness, the prover generates the OR proof π^{OR} , and sends over $(\pi^{\text{OR}}, T, \pi^{\text{NDH}})$.

We prove that the protocol we have just described is adaptive multi-theorem zero-knowledge. Intuitively, this holds since the simulator can fake the proof for the non-DH tuple by running the simulator of Π_{NDH} . Then the proof π^{NDH} can be simulated with respect to a DH tuple, hence any OR proof can be generated using the fact that $T \in L^*$. Given that the OR proof we will use is witness indistinguishable (WI), and that the WI property is maintained under parallel composition, then our final protocol is multi-theorem zero-knowledge. The adaptive zero-knowledge property comes from the fact that the simulator can run internally the simulator of Π_{NDH} to generate the setup (i.e., to program the hash function) without knowing x .

There is a caveat about this protocol. Note that the tuple T can be chosen by the adversarial prover adaptively on the description of the hash function. So, even if we do not need Π_{NDH} to be multi-theorem, it seems that we need it to be at least *adaptive-sound*. To obtain an adaptive-sound NIZK protocol following the FS paradigm, we could rely on the results of [9]. In this, the authors show how to convert any sigma-protocol into an adaptive-sound NIZK protocol using correlation intractable hash functions. However, the Ciampi et al. compiler incurs an efficiency loss, since it requires, for each bit of the challenge of the starting sigma-protocol, to generate two ciphertexts. To avoid this, we first argue that it is sufficient to fix the first two components of the tuple T (g, g^α) in the CRS, and let the adversarial prover choose only X, Y adaptively on the hash function to form the tuple $T = (g, g^\alpha, X, Y)$. We then show how to obtain a protocol Π_{NDH} that remains sound in this *semi-adaptive* adversarial setting, while maintaining reasonable performance (i.e., for a security parameter of 1024 bits the prover and verifier of Π_{NDH} need to perform 50 exponentiations each).

We need to argue that the OR proof also remains sound when part of the tuple T is chosen by the adversary. In the technical part of the paper, we will

show how to realize such an OR proof and provide our new formal definition of soundness that we call *semi-adaptive soundness*, which allows the adversary to decide part of the component of an NP statement. This notion lies in between the standard notion of soundness and the notion of adaptive soundness, which allows the adversary to decide all the parameters of the NP instance to be proven.

On Adaptive Soundness. So far we have mostly focused on obtaining an adaptive zero-knowledge scheme that allows the re-use of the hash-key. We have not mentioned whether it is possible to also prove that our NIZK is adaptive sound. We argue that if the trapdoor sigma-protocol Π_L admits a special type of extractor (in [9] the authors show that any sigma-protocol can be modified to enjoy this special property), then our NIZK is also adaptive-sound. We refer to the technical part of our paper for more detail.

1.3 Related Work

One of the works most related to ours is [7]. In this, the authors construct an adaptive sound, adaptive zero-knowledge, multi-theorem NIZK from correlation intractable hash functions (plus other assumptions like LWEs, or DDH and LPN). However, the results of [7] follow a different spirit compared to ours (and compared also to [4]). As discussed in the previous section, a multi-theorem adaptive NIZK can be trivially obtained using a folklore technique. Namely, it is easy to construct an adaptive multi-theorem NIZK protocol from the same assumptions we use in our paper by following the FLS approach. However, this approach produces a CRS that has two components: a hash key, and a tuple $T \notin L^*$. Hence, the soundness of the protocol depends on T not being in L^* . This is in contrast with what happens in the standard FS transform where the soundness depends only on the soundness of the underlying sigma protocol and on the CI of the hash function. All the multi-theorem protocols proposed in [7] have a similar drawback. That is, the soundness is based on a public key (that is part of the CRS) being sampled correctly. If such a public key is not sampled correctly then the soundness trivially does not hold. In our work, we instead get the same advantage of the FS approach (and of the results proposed in [4]) by providing a protocol whose soundness is based on the correlation intractability of the hash function and on the soundness of the underlying trapdoor sigma-protocol only. To give a concrete example of the benefit of our compiler compared to existing solutions we note the following. If we instantiate our NIZK with the trapdoor sigma-protocol for the language of Diffie-Hellman tuples, we obtain a multi-theorem adaptive NIZK where the CRS consists of the hash key, and two group elements (g, h) . The soundness of this NIZK then holds as long as the hash-key is honestly generated, while (g, h) can be maliciously generated.

Our work follows the spirit of [4], where the authors show how to compile any trapdoor sigma protocol into a NIZK using the FS approach. We extend the approach of Canetti et al. proposing a compiler that turns *any* trapdoor sigma-protocol into a *multi-theorem adaptive* NIZK. Hence, any improvement in

the efficiency of trapdoor sigma protocols has an immediate impact on the performance of our NIZK. [7] follows a different path by proposing ad-hoc schemes that depart from the Fiat-Shamir approach. The advantage of [7] over our work is that the results of [7] are UC secure and tolerate adaptive corruption.

2 Preliminaries

Notations. We denote the security parameter by λ and use “||” as the concatenation operator. For a finite set Q , $x \leftarrow \$ Q$ denotes a sampling of x from Q with uniform distribution. We use “=” to check the equality of two different elements, “ \leftarrow ” as the assigning operator (e.g. to assign to a the value of b we write $a \leftarrow b$). We use the abbreviation PPT which stands for probabilistic polynomial time. We use $\text{poly}(\cdot)$ to indicate a generic polynomial function. We denote with \mathbb{Z}_p the set of integers, where p is the order of the set, with \mathbb{N} the set of natural numbers. We use $G.\text{Gen}(1^\lambda)$ to represent the algorithm to find the generator in the group G . ν represents the negligible function, and δ represents the non-negligible function. For an NP language L we denote the corresponding NP-relation with \mathcal{R}_L . We assume familiarity with the notions of negligible and non-negligible functions, and also the notion of interactive proof systems.

2.1 Diffie-Hellman Related Definitions

Let G be the group of an order p , with a generator g . Let $T = (g, h = g^x, X, Y)$ be a tuple, where $x \in \mathbb{Z}_p$. Let $L_{\text{DH}} = \{T \in G^4 \mid \exists w \in \mathbb{Z}_p : X = g^w \wedge Y = h^w\}$ be the language of DH tuples. Let $L_{\text{NDH}} = \{T \in G^4 \mid \exists w, w' \in \mathbb{Z}_p : X = g^w \wedge Y = h^{w'} \wedge w \neq w'\}$ be the language of non-DH tuples.

We assume the Decisional Diffie-Hellman (DDH) hardness assumption holds in the group G . The DDH hardness assumption is as follows:

Definition 1 (DDH hardness Assumption). *For every PPT algorithm \mathcal{A} :*

$$\left| \Pr[\mathcal{A}(T) = 1 \mid T \in L_{\text{DH}}] - \Pr[\mathcal{A}(T) = 1 \mid T \in L_{\text{NDH}}] \right| \leq \nu(\lambda).$$

2.2 Non-Interactive Argument Systems Related Definitions

We recall the notion of non-interactive argument systems here.

Definition 2 (Non-Interactive Zero-Knowledge Argument Systems).

A non-interactive zero-knowledge argument system (NIZK) for an NP-language L with the corresponding relation R_L is a non-interactive protocol $\Pi = (\text{Setup}, \text{P}, \text{V})$, where:

- $\text{Setup}(1^n, 1^\lambda)$ takes as the input a statement length n and a security parameter λ . It outputs a common reference string crs .
- $\text{P}(\text{crs}, x, w)$ takes as the input crs , the statement x and the witness w , s.t. $(x, w) \in R_L$. It outputs the proof π .

- $V(crs, x, \pi)$ takes as the input crs, x and π . It outputs 1 to accept and 0 to reject.

Π has the following properties:

- **Completeness.** For all $\lambda \in \mathbb{N}$, and all $(x, w) \in R_L$, it holds that:

$$\Pr \left[V(crs, x, P(crs, x, w)) = 1 \mid crs \leftarrow_{\$} \text{Setup}(1^{|x|}, 1^\lambda) \right] = 1 - \nu(\lambda)$$

- **Soundness.** For all PPT provers P^* , s.t. for all $\lambda \in \mathbb{N}$, and all $x \notin L$, it holds that:

$$\Pr \left[V(crs, x, \pi) = 1 \mid crs \leftarrow_{\$} \text{Setup}(1^{|x|}, 1^\lambda); \pi \leftarrow_{\$} P^*(crs) \right] \leq \nu(\lambda).$$

- **Zero knowledge.** There exists a PPT simulator Sim such that for every $(x, w) \in R_L$, the distribution ensembles $\{(crs, \pi) : crs \leftarrow_{\$} \text{Setup}(1^{|x|}, 1^\lambda); \pi \leftarrow_{\$} P(crs, x, w)\}_{\lambda \in \mathbb{N}}$ and $\{\text{Sim}(1^\lambda, x)\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable.

2.3 Sigma-protocol Related Definitions

Most of the following definitions are taken from [4, 10].

Definition 3 (Sigma-protocol). Assuming there is a three-round public-coin interactive protocol $\Sigma = (\text{Gen}, P, V)$ for a NP language L (and corresponding relation R_L) in the common reference string model, where:

- Gen takes as input the unary representation of the security parameter, and it outputs the common reference string crs .
- In the first round of the protocol, P takes as input the common reference string crs , the instance x , the witness w , the randomness R , and it will output the first round message a .
- In the second round, V takes as input the crs, x, a , and it will output the challenge c .
- In the third round, P takes as input the crs, x, w, a, c, R , and it will output the third round message z .
- When V receives (crs, x, a, c, z) as inputs, it outputs 1 to accept and 0 to reject.

Σ is a sigma-protocol if satisfies the following properties:

- **Completeness:** If $(x, w) \in R_L$, then all honest generated transcripts are accepting.
- **Optimal soundness:** For every common reference string crs , every instance $x \notin L$, and every first message a , there is at most one challenge $c = f(crs, x, a)$ such that (crs, x, a, c, z) is an accepting transcript for any choice of third message z . We informally call f the “bad-challenge function” associated with Σ and note that f may not be efficiently computable.

- **Special HVZK:** *There exists a PPT simulator algorithm Sim that takes as $x \in L$ and $c \in \{0, 1\}^\ell$, and outputs an accepting transcript for x where c is the challenge (we denote this action with $(a, z) \leftarrow \text{Sim}(x, c)$). Moreover, for all ℓ -bit strings c , the distribution of the output of the simulator on input (x, c) is computationally indistinguishable from the distribution of the honest generated transcript obtained when \mathbf{V} sends c as the challenge and \mathbf{P} runs on common input x and any private input w such that $(x, w) \in R_L$.*

Remark 1. The Definition 3 is a bit different from the standard notion of sigma-protocol [12] since we only require the protocol to be the optimal sound (instead of special-sound).

Then we recall the definition of the instance-dependant trapdoor sigma-protocol from [4].

Definition 4 (Instance-dependant trapdoor sigma-protocol [4]). *We say that a sigma-protocol $\Sigma = (\text{Gen}, \mathbf{P}, \mathbf{V})$ with bad-challenge function f is an instance-dependant trapdoor sigma-protocol if there are PPT algorithms TrapGen , BadChallenge with the following syntax.*

- $\text{TrapGen}(1^\lambda, x, \text{aux})$ *takes as input the unary representation of the security parameter, an instance x , and an auxiliary input aux . It outputs a common reference string crs along with a trapdoor τ .*
- $\text{BadChallenge}(\tau, \text{crs}, x, a)$ *takes as input a trapdoor τ , common reference string crs , instance x , and first message a . It outputs a challenge c .*

We additionally require the following properties:

- **CRS Indistinguishability:** *For any (x, aux) , an honestly generated common reference string crs is computationally indistinguishable from a common reference string output by $\text{TrapGen}(1^\lambda, x, \text{aux})$.*
- **Correctness:** *For every instance $x \notin L$, there exists an auxiliary input aux such that for all $(\text{crs}, \tau) \leftarrow_{\$} \text{TrapGen}(1^\lambda, x, \text{aux})$, we have that $\text{BadChallenge}(\tau, \text{crs}, x, a) = f(\text{crs}, x, a)$.*

OR Composition of ID Trapdoor Sigma-Protocols. In our paper, we also argue that the OR composition [11] of any 2 instance-dependant trapdoor sigma-protocols (for the relation R_{L_0} and R_{L_1}) is an instance-dependant trapdoor sigma-protocol for the relation $R_{L_0 \vee L_1}$. Moreover, the resulting protocol is witness indistinguishable (WI). In more detail, assuming the existence of 2 instance-dependant trapdoor sigma-protocols $\Sigma_{L_0} = (\text{Gen}_{L_0}, \mathbf{P}_{L_0}, \mathbf{V}_{L_0})$ and $\Sigma_{L_1} = (\text{Gen}_{L_1}, \mathbf{P}_{L_1}, \mathbf{V}_{L_1})$ respectively for the NP language L_0 and L_1 , then the application of the compiler of [11] yields a protocol $\Sigma_{L_0 \vee L_1}$ such that the following lemma holds.

Lemma 1. $\Sigma_{L_0 \vee L_1}$ *is an instance-dependant trapdoor sigma-protocol and it is witness indistinguishable for the language $L = \{(x_0, x_1) : x_0 \in L_0 \vee x_1 \in L_1\}$*

Due to lack of space, we refer the reader to the full version to see full proofs.

2.4 Multi-theorem, Adaptive Non-interactive Proofs

Definition 5 (Adaptive Multi-Theorem Zero Knowledge [17]). *Assuming we have a non-interactive protocol $\Pi = (\text{Setup}, \text{P}, \text{V})$ for an NP language L with corresponding relation R_L . Π is adaptive multi-theorem zero knowledge if for any PPT algorithm \mathcal{A} , there exists a PPT simulator $\text{Sim} = (\text{Sim}_0, \text{Sim}_1)$, running in (expected) polynomial time, such that for polynomial bounded q :*

$$\left| \Pr \left[\text{Expt}_{\Pi, \text{Sim}, \mathcal{A}}(1^{|x|}, 1^\lambda) = 1 \right] - \frac{1}{2} \right| \leq \nu(\lambda)$$

The experiment $\text{Expt}_{\Pi, \text{Sim}, \mathcal{A}}(1^{|x|}, 1^\lambda)$ is defined as follows:

$\text{Expt}_{\Pi, \text{Sim}, \mathcal{A}}(1^{|x|}, 1^\lambda)$:

$b \leftarrow_{\$} \{0, 1\}, q \leftarrow 0, \text{State}_{\mathcal{A}} \leftarrow \emptyset, \text{crs}_0 \leftarrow_{\$} \text{Setup}(1^{|x|}, 1^\lambda), (\text{crs}_1, \tau_{\text{Sim}_1}) \leftarrow_{\$} \text{Sim}_0(1^{|x|}, 1^\lambda)$

repeat

$q \leftarrow q + 1, (\text{State}_{\mathcal{A}}, x, w) \leftarrow_{\$} \mathcal{A}(1^\lambda, \text{crs}_b, \text{State}_{\mathcal{A}})$

if $(x, w) \in R_L$ then $\pi_0 \leftarrow_{\$} \text{P}(\text{crs}_0, x, w), \pi_1 \leftarrow_{\$} \text{Sim}_1(\text{crs}_1, \tau_{\text{Sim}_1}, x)$

else $\pi_0 \leftarrow \pi_1 \leftarrow \emptyset$

$(\text{State}_{\mathcal{A}}, \text{cont}, d) \leftarrow_{\$} \mathcal{A}(1^\lambda, \text{State}_{\mathcal{A}}, \pi_b)$

until $\text{cont} = \text{false}$

return $b = d$

Definition 6 (Witness Indistinguishability). *Assuming we have an interactive protocol $\Sigma_L = (\text{Gen}_L, \text{P}_L, \text{V}_L)$ for NP language L . Σ_L is Witness Indistinguishable for relation R_L if, every malicious verifier V_L^* , s.t. for all x, w, w' with $(x, w) \in R_L$ and $(x, w') \in R_L$, it holds that:*

$$\left| \Pr \left[\text{V}_L^*(x, \pi_0) = 1 \mid \pi_0 \leftarrow_{\$} \text{P}_L(x, w) \right] - \Pr \left[\text{V}_L^*(x, \pi_1) = 1 \mid \pi_1 \leftarrow_{\$} \text{P}_L(x, w') \right] \right| \leq \nu(\lambda)$$

2.5 Semi-adaptive Soundness

We now introduce a new notion of soundness that we call *semi-adaptive soundness*. Informally, we see every theorem x as divided into two parts (α, β) , and we require the adversary to specify α before the sampling of the CRS, whereas β can be adaptively chosen from the adversary. More formally:

Definition 7 (Semi-Adaptive Soundness). *Given 2 sets $S_1 \subseteq \{0, 1\}^*, S_2 \subseteq \{0, 1\}^*$, and the NP language $L = \{(\alpha, \beta) \mid \alpha \in S_1 \wedge \beta \in S_2 \wedge \phi(\alpha, \beta) = 1\}$ defined over some predicate ϕ . Assuming we have a non-interactive protocol $\Pi = (\text{Setup}, \text{P}, \text{V})$ for an NP language L with corresponding relation R_L . Π is semi-adaptive sound if for any $\alpha \in S_1$ and for any PPT prover P^* , it holds that:*

$$Pr_{\alpha} \left[(\alpha, \beta) \notin L \wedge \mathbb{V}(crs, (\alpha, \beta), \pi) = 1 \mid \alpha \in S_1 \wedge \beta \in S_2; \right. \\ \left. crs \leftarrow \text{\$ Setup}(1^{|x|}, 1^\lambda); (\pi, \beta) \leftarrow P^*(crs, \alpha) \right] \leq \nu(\lambda).$$

2.6 Semi-instance-dependant (SID) Trapdoor Sigma-Protocol

We introduce an extension of the notion of trapdoor sigma-protocols we denote as *semi-instance-dependant trapdoor sigma-protocol*. Informally, similar to semi-adaptive soundness defined above, we divided every theorem x into 2 parts (α, β) , and the TrapGen and BadChallenge algorithms of the semi-instance-dependant trapdoor sigma-protocol will take α other than the whole theorem x .

Definition 8 (Semi-instance-dependant trapdoor sigma-protocol). Given $S_1 \subseteq \{0, 1\}^*$, $S_2 \subseteq \{0, 1\}^*$, and the NP language $L = \{(\alpha, \beta) \mid \alpha \in S_1 \wedge \beta \in S_2 \wedge \phi(\alpha, \beta) = 1\}$ defined over some predicate ϕ . We say that a sigma-protocol $\Sigma = (\text{Gen}, P, \mathbb{V})$ with bad-challenge function f is a semi-instance-dependant trapdoor sigma-protocol if there are PPT algorithms TrapGen, BadChallenge with the following syntax.

- TrapGen($1^\lambda, \alpha, aux$) takes as input the unary representation of the security parameter, the first part of the instance α , and an auxiliary input aux . It outputs a common reference string crs along with a trapdoor τ .
- BadChallenge(τ, crs, α, a) takes as input a trapdoor τ , common reference string crs , the first part of the instance α , and first message a . It outputs a challenge c .

We additionally require the following properties:

- **CRS Indistinguishability:** For any (α, aux) , an honestly generated common reference string crs is computationally indistinguishable from a common reference string output by TrapGen($1^\lambda, \alpha, aux$).
- **Correctness:** For every instance $x \notin L$, there exists an auxiliary input aux such that for all $(crs, \tau) \leftarrow \text{\$ TrapGen}(1^\lambda, \alpha, aux)$, we have that BadChallenge(τ, crs, α, a) = $f(crs, x, a)$.

We argue that the OR composition of [11] applied on a SID trapdoor sigma-protocol and an ID trapdoor sigma-protocol yields a new SID for the OR relation. More formally, assuming the existence of an ID trapdoor sigma-protocol $\Sigma_{L_0} = (\text{Gen}_{L_0}, P_{L_0}, \mathbb{V}_{L_0})$ for NP language L_0 and a SID trapdoor sigma-protocol $\Sigma_{L_1} = (\text{Gen}_{L_1}, P_{L_1}, \mathbb{V}_{L_1})$ for NP language $L_1 = \{(\alpha, \beta) \mid \alpha \in S_1 \wedge \beta \in S_2 \wedge \phi(\alpha, \beta) = 1\}$, then the application of the compiler of [11] on Σ_{L_0} and Σ_{L_1} will yield a SID trapdoor sigma-protocol $\Sigma_{L_0 \vee L_1}$, such that the following lemma holds.

Lemma 2. $\Sigma_{L_0 \vee L_1}$ is a semi-instance-dependant trapdoor sigma-protocol, and it is witness indistinguishable, for NP language $L = \{((\alpha, x), \beta) \mid (\alpha, x) \in S'_1 \wedge \beta \in S'_2 \wedge (\phi(\alpha, \beta) = 1 \vee x \in L_0)\}$, where $S'_1 = S_1 \times \{0, 1\}^*$ and $S'_2 = S_2$.

Proof. The proof is nearly identical to the proof for Lemma 1.

2.7 Correlation-intractable Hash Functions and FS Transform

Here we recall the related definitions of Correlation-Intractable Hash Family (CIHF) from [4].

Definition 9 (Hash family). For a pair of efficiently computable functions $(n(\cdot), m(\cdot))$, a hash family with input length n and output length m is a collection $\mathcal{H} = \{h_k : \{0, 1\}^{n(\lambda)} \rightarrow \{0, 1\}^{m(\lambda)}\}_{\lambda \in \mathbb{N}, k \in \{0, 1\}^{s(\lambda)}}$ of keyed hash functions, along with a pair of PPT algorithms specified as follows: (i) $\mathcal{H}.\text{Gen}(1^\lambda)$ outputs a hash key $k \in \{0, 1\}^{s(\lambda)}$; (ii) $\mathcal{H}.\text{H}(k, x)$ computes the function $h_k(x)$.

Definition 10 (Correlation intractability). For a given relation ensemble $R := \{R_\lambda \subseteq \{0, 1\}^{n(\lambda)} \times \{0, 1\}^{m(\lambda)}\}$, a hash family $\mathcal{H} = \{h_k : \{0, 1\}^{n(\lambda)} \rightarrow \{0, 1\}^{m(\lambda)}\}_{\lambda \in \mathbb{N}, k \in \{0, 1\}^{s(\lambda)}}$ is said to be R -correlation intractable with security (σ, δ) if for every σ -size attacker $\mathcal{A} := \{\mathcal{A}_\lambda\}$:

$$\Pr[(x, h_k(x)) \in R_\lambda : k \leftarrow \mathcal{H}.\text{Gen}(1^\lambda); x \leftarrow \mathcal{A}(k)] = O(\delta(\lambda)).$$

We say that \mathcal{H} is R -correlation intractable if it is R -correlation intractable with security $(\lambda^c, \lambda^{-c})$ for all constants $c > 1$.

Definition 11 (Sparsity). For any relation ensemble $R := \{R_\lambda \subseteq \{0, 1\}^{n(\lambda)} \times \{0, 1\}^{m(\lambda)}\}_\lambda$, we say that R is $\rho(\cdot)$ -sparse if for all $\lambda \in \mathbb{N}$ and for any $x \in \{0, 1\}^{n(\lambda)}$ it holds that $(x, y) \in R_\lambda$ with probability at most $\rho(\lambda)$ over the choice of $y \leftarrow \{0, 1\}^{m(\lambda)}$. When ρ is a negligible function, we say that R is sparse.

Efficiently Searchable Relations. In this work, we will need hash families to achieve correlation intractability for relations R with a unique output $y = f(x)$ associated to each input x , and such that $y = f(x)$ is an efficiently computable function of x .

Definition 12 (Unique output relation). We say that a relation R is a unique output relation if for every input x , there exists at most one output y such that $(x, y) \in R$.

Definition 13 (Efficiently searchable relation). We say that a (necessarily unique-output) relation ensemble R is searchable in (non-uniform) time t if there exists a function $f = f_R : \{0, 1\}^* \rightarrow \{0, 1\}^*$ computable in (non-uniform) time t such that for any input x , if $(x, y) \in R$ then $y = f(x)$; that is, $f(x)$ is the unique y such that $(x, y) \in R$, provided that such a y exists. We say that R is efficiently searchable if it is searchable in time $\text{poly}(n)$.

Programmability. The following property turns out to be very useful to prove the zero-knowledge property of non-interactive proofs derived using correlation intractable hash families.

Definition 14 (1-universality). We say that a hash family \mathcal{H} is 1-universal if for any $\lambda \in \mathbb{N}$, input $x \in \{0, 1\}^{n(\lambda)}$, and output $y \in \{0, 1\}^{m(\lambda)}$, we have $\Pr[h_k(x) = y : k \leftarrow_s \mathcal{H}.Gen(1^\lambda)] = 2^{-m(\lambda)}$.

We say that a hash family \mathcal{H} is programmable if it is 1-universal, and if there exists an efficient sampling algorithm $Samp(1^\lambda, x, y)$ that samples from the conditional distribution $k \leftarrow_s \mathcal{H}.Gen(1^\lambda) | h_k(x) = y$.

We recall the theorem from [4] that we use in our work:

Theorem 1 ([4]). Suppose that \mathcal{H} is a hash family that is correlation-intractable for all subexponentially sparse relations that are searchable in time T . Moreover, suppose that $\Sigma = (Gen, P, V, TrapGen, BadChallenge)$ is an instance-dependent trapdoor sigma-protocol with $2^{-\lambda^\epsilon}$ soundness for some $\epsilon > 0$, such that $BadChallenge(\tau, crs, x, a)$ is computable in time T . Then, \mathcal{H} soundly instantiates the Fiat-Shamir heuristic for Σ .

A Note on NIZK from ID Trapdoor Sigma-Protocol. Assuming the existence of an ID trapdoor sigma-protocol Σ_L for NP language L , then the application of Theorem 1 on Σ_L will yield a sound NIZK protocol Π_L .

In our work, we also make use of the following lemmas. The application of Theorem 1 on $\Sigma_{L_0 \vee L_1}$ (from Lemma 1) will yield a NIZK protocol $\Pi_{L_0 \vee L_1}$, such that the following lemma holds.

Lemma 3. $\Pi_{L_0 \vee L_1}$ is sound and WI.

For Lemma 4, it states that FS transform with CIHF applied on any SID trapdoor sigma-protocols will yield a semi-adaptive sound NIZK.

Lemma 4. Let Σ_L be a semi-instance-dependant trapdoor sigma-protocol, for language $L = \{(\alpha, \beta) \mid \alpha \in S_\alpha \wedge \beta \in S_\beta \wedge \phi(\alpha, \beta) = 1\}$. Then, NIZK Π_L obtained by applying FS transform with a CIHF \mathcal{H} on Σ_L , is semi-adaptive sound, for language L .

We refer to the full version for the formal argument.

The Existence of the SID Trapdoor Sigma-Protocols. In [9] the authors observe that it is possible to extract the unique bad-challenge for well-known Chaum-Pedersen sigma-protocols [8] for DH tuples that we denote with Σ_{DH} (we recall it in Fig. 1, where $crs = \emptyset$).

In particular, the authors show how to extract the bad-challenge of the 1-bit challenge version of the sigma-protocol Σ_{DH} for DH tuples. We show that the parallel repetition version Σ_{DH}^t is a SID trapdoor sigma-protocol for $L_{DH} = \{(g, h, X, Y) \mid (g, h) \in S_1 \wedge (X, Y) \in S_2 \wedge \phi(g, h, X, Y) = 1\}$, where $S_1 = \{(g, g^x) \in G \times G \mid x \in \mathbb{Z}_p\}$, $S_2 = \{(h, h^y) \in G \times G \mid y \in \mathbb{Z}_p\}$, and $\phi(g, h, X, Y) = 1$ if and only if $\exists w \in \mathbb{Z}_p : X = g^w \wedge Y = h^w$. Formally (proofs are in the full version):

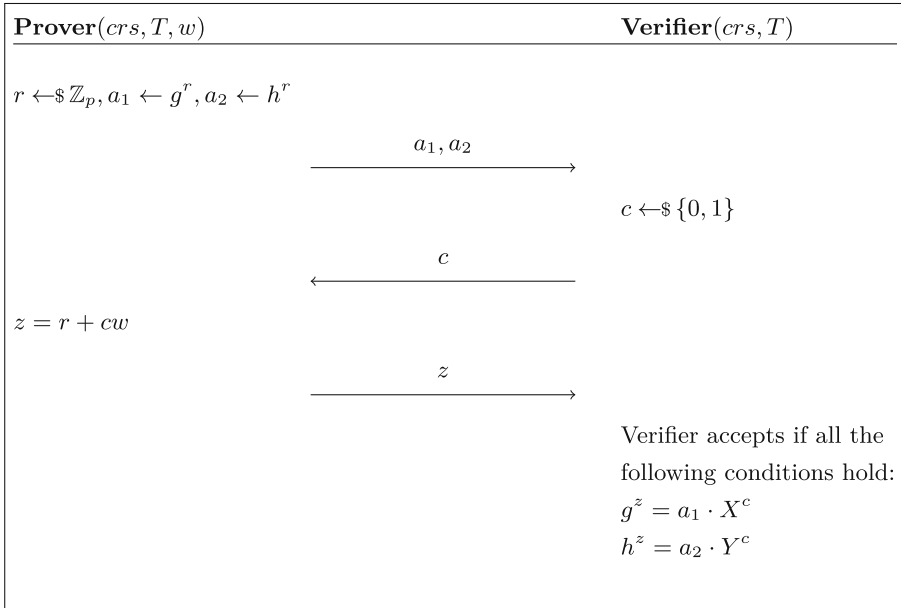


Fig. 1. Sigma-protocol Σ_{DH} for L_{DH}

Theorem 2. Let Σ_{DH}^t be the parallel repetition version of Σ_{DH} , with the number of repetition t . Then, Σ_{DH}^t is a semi-instance-dependant trapdoor sigma-protocol, for L_{DH} .

One of the main tools we rely on is a SID trapdoor sigma-protocol for the language of the non-DH tuple. In particular, we need to construct a protocol Σ_{NDH} for the language $L_{\text{NDH}} = \{(g, h, X, Y) \mid (g, h) \in S_1 \wedge (X, Y) \in S_2 \wedge \phi(g, h, X, Y) = 1\}$, where $S_1 = \{(g, g^x) \in G \times G \mid x \in \mathbb{Z}_p\}$, $S_2 = \{(h, h^y) \in G \times G \mid y \in \mathbb{Z}_p\}$, and $\phi(g, h, X, Y) = 1$ if and only if $\exists w, w' \in \mathbb{Z}_p : X = g^w \wedge Y = h^{w'} \wedge w \neq w'$. At a high level, our protocol works as follows. The prover computes a commitment of a random value $b \in \{0, 1\}^\tau$. The commitment is equivocal when $T \in L_{\text{NDH}}$ and it is binding (and extractable) otherwise. The prover sends the commitment of b to the verifier, who replies with a uniformly random $c \in \{0, 1\}$. In the third round, the prover will equivocate the commitment to an opening of c , and send the opening information to the verifier. We recall that the honest prover can always equivocate the commitment since $T \in L_{\text{NDH}}$.

This protocol is sound since when $T \notin L_{\text{NDH}}$, the probability of the prover providing a valid opening for c is $2^{-\tau}$. To extract the bad-challenge, we will rely on the fact that the commitment is extractable when $T \notin L_{\text{NDH}}$. In particular, we prove that it is possible to extract the bad-challenge for a proof computed with respect to a tuple $T = (g, h, X, Y)$, having access only to the discrete logarithm of h . This is the reason why our protocol is only semi-adaptive and not fully adaptive (i.e., if the entire tuple was chosen by the adversary then the extractor

would have no access to the discrete logarithm of h). We refer the reader to the Appendix A for details of this concrete construction.

One nice feature of the protocol we have described is that for a challenge of size $\tau = \log \lambda$, where λ is the security parameter, prover and verifier need to perform only 5 exponentiations each (We refer to Appendix A.1 for the efficiency analysis). We see Σ_{NDH} as a result of independent interest. Previous to our work, it was already known how to construct a trapdoor sigma protocol with similar performance, but ours is the first protocol to have such performance while being a SID trapdoor sigma-protocol. In particular, we note that in [21], the authors give a construction of trapdoor sigma-protocol for the language of DH (hence, also for the language of non-DH) tuples with similar performance as ours. Unfortunately, it is not clear how to prove that the protocol proposed in [21] is also a SID trapdoor sigma-protocols.

3 NIZK with Adaptive Multi-theorem ZK

In this section, we show how to obtain our adaptive multi-theorem ZK and sound NIZK protocol for an NP language L , assuming that we have an ID trapdoor sigma-protocol $\Sigma_L = (\text{Gen}_L, \text{P}_L, \text{V}_L)$ for L . For our construction we make use of the following tools:

- A hash family \mathcal{H} that is correlation-intractable for all subexponentially sparse relations that are searchable in time T , which is also programmable.
- The SID trapdoor sigma-protocol $\Sigma_{\text{OR}} = (\text{Gen}_{\text{OR}}, \text{P}_{\text{OR}}, \text{V}_{\text{OR}})$ of Sect. 2.6 for NP language $L_{\text{OR}} = L \vee L_{\text{DH}} = \{((g, h, x), (X, Y)) \mid (g, h, x) \in S_1 \wedge (X, Y) \in S_2 \wedge (\phi(g, h, X, Y) = 1 \vee x \in L)\}$, where $S_1 = \{(g, g^\alpha, x) \in G \times G \times \{0, 1\}^* \mid \alpha \in \mathbb{Z}_p\}$, $S_2 = \{(h, h^\beta) \in G \times G \mid \beta \in \mathbb{Z}_p\}$, and $\phi(g, h, X, Y) = 1$ if and only if $\exists w \in \mathbb{Z}_p : X = g^w \wedge Y = h^w$. The protocol Σ_{OR} has $2^{-\lambda^\epsilon}$ soundness for $\epsilon > 0$. We note that this protocol can be obtained starting from Σ_L and any SID trapdoor sigma protocol Σ_{DH} for L_{DH} . We provide an example (Σ_{DH}^t from Theorem 2) to be used as Σ_{DH} .
- A SID trapdoor sigma-protocol $\Sigma_{\text{NDH}} = (\text{Gen}_{\text{NDH}}, \text{P}_{\text{NDH}}, \text{V}_{\text{NDH}})$ for L_{NDH} . Σ_{NDH} need to have $2^{-\lambda^\epsilon}$ soundness for $\epsilon > 0$.

We denote the obtained NIZK protocol with $\Pi = (\text{Setup}, \text{P}, \text{V})$. The Setup algorithm works as follows:

- $crs_L \leftarrow \text{Gen}_L(1^\lambda)$, $crs_{\text{DH}} \leftarrow crs_{\text{NDH}} \leftarrow \emptyset$, $g \leftarrow G.\text{Gen}(1^\lambda)$, $x \leftarrow \mathbb{Z}_p$, $h \leftarrow g^x$, $k \leftarrow \mathcal{H}.\text{Gen}(1^\lambda)$.
- output $(crs_L, crs_{\text{DH}}, crs_{\text{NDH}}, (g, h), k)$

We formally describe the interaction between the prover and the verifier of Π in Fig. 2.

Before proving the security of Π , we need to prove that the FS transform applied on Σ_{OR} yields a WI semi-adaptive sound non-interactive protocol. This comes immediately from Lemma 2, 3, and 4. Hence, if we denote with Π_{OR} the non-interactive protocol resulting from the application of the FS transform on Σ_{OR} we can claim the following.

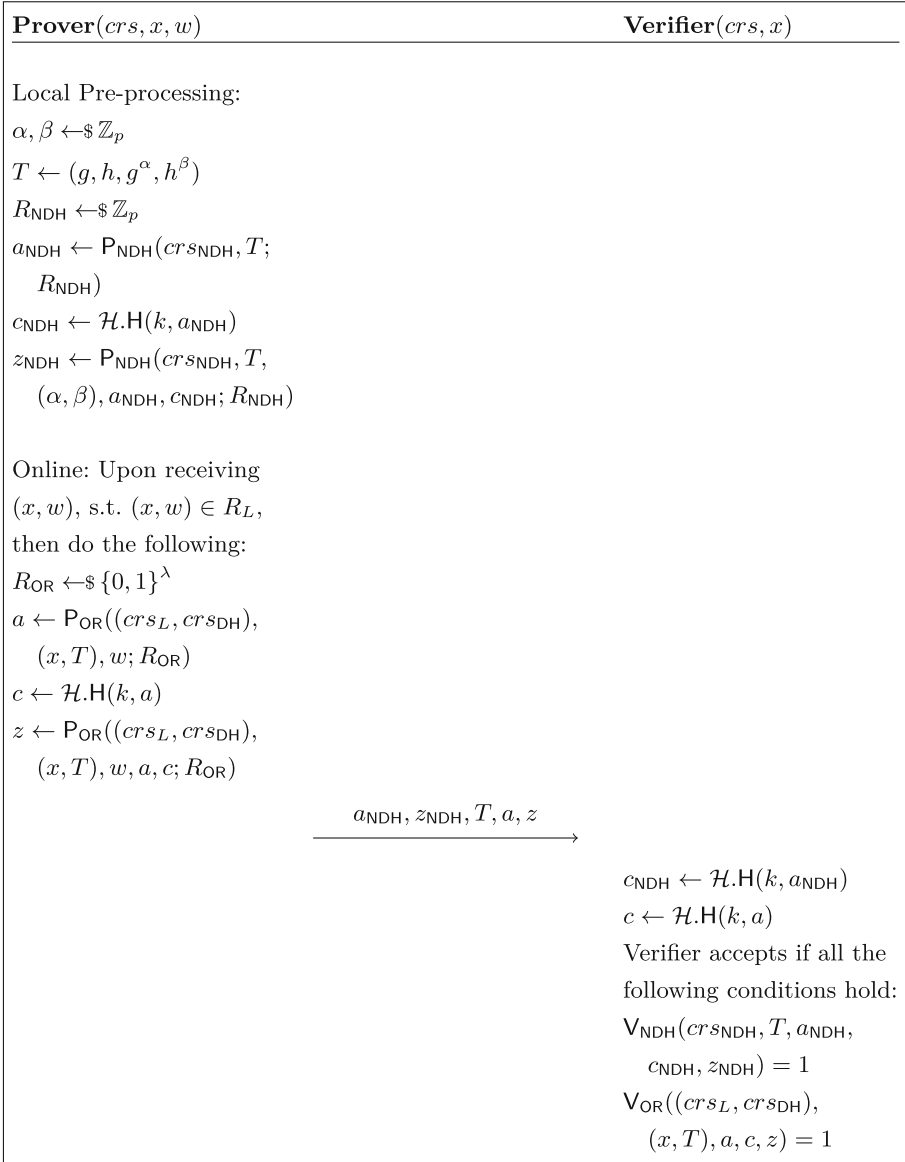


Fig. 2. Our NIZK protocol Π

Theorem 3. Π_{OR} is WI semi-adaptive sound for L_{OR} .

We are now ready to prove our main lemmas.

Lemma 5. Let Π be the protocol of Fig. 2, then Π is sound.

Proof. Assuming Π is not sound, then there exists a PPT algorithm \mathcal{A} , s.t.:

$$\Pr_x \left[x \notin L \wedge \mathsf{V}(crs, x, \pi) = 1 \mid crs \leftarrow \$ \mathsf{Setup}(1^{|x|}, 1^\lambda); \pi \leftarrow \$ \mathcal{A}(crs, x) \right] \geq \delta(\lambda).$$

To make V accept when $x \notin L$, there are 2 possibilities:

- When T is a DH tuple, $\mathsf{V}_{\text{NDH}}(crs_{\text{NDH}}, T, a_{\text{NDH}}, c_{\text{NDH}}, z_{\text{NDH}}) = 1$, and $\mathsf{V}_{\text{OR}}((crs_L, crs_{\text{DH}}), (x, T), a, c, z) = 1$. If V_{NDH} accepts when $T \notin L_{\text{NDH}}$ then it means that \mathcal{A} can find $(a_{\text{NDH}}, c_{\text{NDH}}, z_{\text{NDH}})$ to make $(crs_{\text{NDH}}, T, a_{\text{NDH}}, c_{\text{NDH}}, z_{\text{NDH}})$ accepting with non-negligible probability, and it directly contradicts to the semi-adaptive soundness of Π_{NDH} . Formally, we can construct the following adversary \mathcal{A}' :

$\mathcal{A}'(crs_{\text{NDH}}, k, \alpha)$:

$crs_L \leftarrow \$ \mathsf{Gen}_L(1^\lambda)$, $crs_{\text{DH}} \leftarrow \emptyset$, parsing α as (g, h) , $w \leftarrow \$ \mathbb{Z}_p$, $\beta \leftarrow (g^w, h^w)$,
 $x \leftarrow (g, h, \beta)$, $crs \leftarrow (crs_L, crs_{\text{DH}}, crs_{\text{NDH}}, (g, h), k)$
 waiting for receiving all π from $\mathcal{A}(crs, x)$
return all (π, β)

Now we have the following observation: 1) \mathcal{A} works correctly. We know $crs_L \leftarrow \$ \mathsf{Gen}_L(1^\lambda)$, $crs_{\text{DH}} = \emptyset$. Also, the hash key k , crs_{NDH} and (g, h) are provided by the challenger, so we can conclude that crs is the same as $crs \leftarrow \$ \mathsf{Setup}(1^{|x|}, 1^\lambda)$. 2) The output of \mathcal{A} makes V accept with non-negligible probability, and it means that we find an accepting proof π when $(\alpha, \beta) \notin L_{\text{NDH}}$. This contradicts Lemma 4.

- When T is a non-DH tuple, $\mathsf{V}_{\text{NDH}}(crs, T, a_{\text{NDH}}, c_{\text{NDH}}, z_{\text{NDH}}) = 1$, and $\mathsf{V}_{\text{OR}}(crs, (x, T), a, c, z) = 1$. Then V_{NDH} accepts because $T \in L_{\text{NDH}}$. However, if V_{OR} accepts when $x \notin L \wedge T \notin L_{\text{DH}}$ it means that the adversary \mathcal{A} is able to find (a, c, z) to make $(crs, (x, T), a, c, z)$ accepting with non-negligible probability, and it directly contradicts the semi-adaptive soundness of Π_{OR} . The reduction is identical to the reduction for Π_{NDH} above, and it contradicts Theorem 3.

We note that in this proof, the security only relies on the soundness of Π_{NDH} and Π_{OR} , where their soundness relies on the CI property of CIHF. We do not use the DDH assumption here. \square

Lemma 6. *Let Π be the protocol of Fig. 2, then Π is adaptive multi-theorem zero-knowledge.*

Proof. We have the following simulator $\text{Sim} = (\text{Sim}_0, \text{Sim}_1)$, by having SHVZK simulator Sim_{NDH} from Σ_{NDH} :

$\text{Sim}(1^{|x|}, 1^\lambda) :$

```

 $crs_L \leftarrow \text{Gen}(1^\lambda), crs_{DH} \leftarrow crs_{SNDH} \leftarrow \emptyset, g \leftarrow G.\text{Gen}(1^\lambda), x, w_{DH} \leftarrow \mathbb{Z}_p, c_{NDH}$ 
 $\leftarrow \{0, 1\}^\lambda$ 
 $h \leftarrow g^x, T_{DH} \leftarrow (g, h, g^{w_{DH}}, h^{w_{DH}}), (a_{NDH}, z_{NDH}) \leftarrow \text{Sim}_{NDH}(T_{DH}, c_{NDH})$ 
 $k \leftarrow \text{Samp}(1^\lambda, a_{NDH}, c_{NDH}), crs \leftarrow (crs_L, crs_{DH}, crs_{SNDH}, (g, h), k)$ 
 $\tau_{\text{Sim}} \leftarrow (T_{DH}, w_{DH}, a_{NDH}, z_{NDH})$ 
return  $crs, \tau_{\text{Sim}}$ 

```

$\text{Sim}(crs, \tau_{\text{Sim}}, x) :$

```

 $R_{OR} \leftarrow \{0, 1\}^\lambda, a \leftarrow \text{P}_{OR}((crs_L, crs_{DH}), (x, T_{DH}), w_{DH}; R_{OR}), c \leftarrow \mathcal{H}.H(k, a)$ 
 $z \leftarrow \text{P}_{OR}((crs_L, crs_{DH}), (x, T_{DH}), w_{DH}, a, c; R_{OR})$ 
return  $(a_{NDH}, z_{NDH}, T_{DH}, a, z)$ 

```

We prove this lemma through hybrid experiments. We denote the output of adversary in the hybrid \mathcal{H}_i with $out^{\mathcal{H}_i}$, to show for $i = \{0, 1, 2, 3\}$: $|\Pr[out^{\mathcal{H}_i}] - \Pr[out^{\mathcal{H}_{i+1}}]| \leq \nu(\lambda)$. We note that $out^{\mathcal{H}_0}$ corresponds to the output of the adversary in the real game, and $out^{\mathcal{H}_4}$ corresponds to the output of the adversary in the simulated experiments. We highlight the part that has differences for better understanding:

<p>$\mathcal{H}_0 :$</p> <pre> $State_A \leftarrow \emptyset$ $crs_L \leftarrow \text{Gen}_L(1^\lambda), crs_{SNDH} \leftarrow crs_{DH} \leftarrow \emptyset, g \leftarrow G.\text{Gen}(1^\lambda)$ $x, \alpha, \beta, R_{NDH} \leftarrow \mathbb{Z}_p$ $h \leftarrow g^x, k \leftarrow \mathcal{H}.\text{Gen}(1^\lambda)$ $crs \leftarrow (crs_L, crs_{DH}, crs_{SNDH}, (g, h), k)$ $T \leftarrow (g, h, g^\alpha, h^\beta)$ $a_{NDH} \leftarrow \text{P}_{NDH}(crs_{SNDH}, T; R_{NDH})$ $c_{NDH} \leftarrow \mathcal{H}.H(k, a_{NDH})$ $z_{NDH} \leftarrow \text{P}_{NDH}(crs_{SNDH}, T, (\alpha, \beta), a_{NDH}, c_{NDH}; R_{NDH}); \text{repeat}$ $(State_A, x, w) \leftarrow \mathcal{A}(1^\lambda, crs, State_A)$ if $(x, w) \in R_L$ then $R_{OR} \leftarrow \{0, 1\}^\lambda, a \leftarrow \text{P}_{OR}((crs_L, crs_{DH}), (x, T), w; R_{OR})$ $c \leftarrow \mathcal{H}.H(k, a), z \leftarrow \text{P}_{OR}((crs_L, crs_{DH}), (x, T), w, a, c; R_{OR})$ $\pi \leftarrow (a_{NDH}, z_{NDH}, T, a, z)$ else $\pi \leftarrow \emptyset$ $(State_A, cont, d) \leftarrow \mathcal{A}(1^\lambda, State_A, \pi)$ until $cont = \text{false}$ return $d = 0$ </pre>

$\mathcal{H}_1 : State_A \leftarrow \emptyset, crs_L \leftarrow \text{Gen}_L(1^\lambda), crs_{NDH} \leftarrow crs_{SDH} \leftarrow \emptyset, g \leftarrow G.\text{Gen}(1^\lambda)$
 $x, \alpha, \beta, R_{NDH} \leftarrow \mathbb{Z}_p, c_{NDH} \leftarrow \{0, 1\}^\lambda, h \leftarrow g^x, T \leftarrow (g, h, g^\alpha, h^\beta)$
 $a_{NDH} \leftarrow P_{NDH}(crs_{NDH}, T; R_{NDH}), k \leftarrow \text{Samp}(1^\lambda, a_{NDH}, c_{NDH})$
 $crs \leftarrow (crs_L, crs_{SDH}, crs_{NDH}, (g, h), k), z_{NDH} \leftarrow P_{NDH}(crs_{NDH}, T, (\alpha, \beta), a_{NDH}, c_{NDH}; R_{NDH})$
 repeat
 $(State_A, x, w) \leftarrow \mathcal{A}(1^\lambda, crs, State_A)$
 if $(x, w) \in R_L$ then $R_{OR} \leftarrow \{0, 1\}^\lambda, a \leftarrow P_{OR}((crs_L, crs_{SDH}), (x, T), w; R_{OR})$
 $c \leftarrow \mathcal{H}.H(k, a), z \leftarrow P_{OR}((crs_L, crs_{SDH}), (x, T), w, a, c; R_{OR})$
 $\pi \leftarrow (a_{NDH}, z_{NDH}, T, a, z)$
 else $\pi \leftarrow \emptyset$
 $(State_A, cont, d) \leftarrow \mathcal{A}(1^\lambda, State_A, \pi)$
 until $cont = \text{false}$
return $d = 0$

$\mathcal{H}_2 : State_A \leftarrow \emptyset, crs_L \leftarrow \text{Gen}_L(1^\lambda), crs_{NDH} \leftarrow crs_{SDH} \leftarrow \emptyset, g \leftarrow G.\text{Gen}(1^\lambda)$
 $x, \alpha, \beta, R_{NDH} \leftarrow \mathbb{Z}_p, c_{NDH} \leftarrow \{0, 1\}^\lambda, h \leftarrow g^x, T \leftarrow (g, h, g^\alpha, h^\beta)$
 $(a_{NDH}, z_{NDH}) \leftarrow \text{Sim}_{NDH}(T, c_{NDH}), k \leftarrow \text{Samp}(1^\lambda, a_{NDH}, c_{NDH})$
 $crs \leftarrow (crs_L, crs_{SDH}, crs_{NDH}, (g, h), k);$ repeat
 $(State_A, x, w) \leftarrow \mathcal{A}(1^\lambda, crs, State_A)$
 if $(x, w) \in R_L$ then $R_{OR} \leftarrow \{0, 1\}^\lambda, a \leftarrow P_{OR}((crs_L, crs_{SDH}), (x, T), w; R_{OR})$
 $c \leftarrow \mathcal{H}.H(k, a), z \leftarrow P_{OR}((crs_L, crs_{SDH}), (x, T), w, a, c; R_{OR})$
 $\pi \leftarrow (a_{NDH}, z_{NDH}, T, a, z)$
 else $\pi \leftarrow \emptyset$
 $(State_A, cont, d) \leftarrow \mathcal{A}(1^\lambda, State_A, \pi)$
 until $cont = \text{false}$
return $d = 0$

$\mathcal{H}_3 : State_A \leftarrow \emptyset, crs_L \leftarrow \text{Gen}_L(1^\lambda), crs_{NDH} \leftarrow crs_{SDH} \leftarrow \emptyset, g \leftarrow G.\text{Gen}(1^\lambda)$
 $x, w_{DH}, R_{NDH} \leftarrow \mathbb{Z}_p, c_{NDH} \leftarrow \{0, 1\}^\lambda, h \leftarrow g^x, T_{DH} \leftarrow (g, h, g^{w_{DH}}, h^{w_{DH}})$
 $(a_{NDH}, z_{NDH}) \leftarrow \text{Sim}_{NDH}(T_{DH}, c_{NDH}), k \leftarrow \text{Samp}(1^\lambda, a_{NDH}, c_{NDH})$
 $crs \leftarrow (crs_L, crs_{SDH}, crs_{NDH}, (g, h), k);$ repeat
 $(State_A, x, w) \leftarrow \mathcal{A}(1^\lambda, crs, State_A)$
 if $(x, w) \in R_L$ then $R_{OR} \leftarrow \{0, 1\}^\lambda, a \leftarrow P_{OR}((crs_L, crs_{SDH}), (x, T_{DH}), w; R_{OR})$
 $c \leftarrow \mathcal{H}.H(k, a), z \leftarrow P_{OR}((crs_L, crs_{SDH}), (x, T_{DH}), w, a, c; R_{OR})$
 $\pi \leftarrow (a_{NDH}, z_{NDH}, T_{DH}, a, z)$
 else $\pi \leftarrow \emptyset$
 $(State_A, cont, d) \leftarrow \mathcal{A}(1^\lambda, State_A, \pi)$
 until $cont = \text{false}$
return $d = 0$

```

 $\mathcal{H}_4 : State_A \leftarrow \emptyset, crs_L \leftarrow \text{Gen}_L(1^\lambda), crs_{NDH} \leftarrow crs_{DH} \leftarrow \emptyset, g \leftarrow G.Gen(1^\lambda)$ 
 $x, w_{DH}, R_{NDH} \leftarrow \mathbb{Z}_p, c_{NDH} \leftarrow \{0, 1\}^\lambda, h \leftarrow g^x, T_{DH} \leftarrow (g, h, g^{w_{DH}}, h^{w_{DH}})$ 
 $(a_{NDH}, z_{NDH}) \leftarrow \text{Sim}_{NDH}(T_{DH}, c_{NDH}), k \leftarrow \text{Samp}(1^\lambda, a_{NDH}, c_{NDH})$ 
 $crs \leftarrow (crs_L, crs_{DH}, crs_{NDH}, (g, h), k); \text{repeat}$ 
   $(State_A, x, w) \leftarrow \mathcal{A}(1^\lambda, crs, State_A)$ 
  if  $(x, w) \in R_L$  then  $R_{OR} \leftarrow \{0, 1\}^\lambda, a \leftarrow \text{P}_{OR}((crs_L, crs_{DH}), (x, T_{DH}), w_{DH}; R_{OR})$ 
     $c \leftarrow \mathcal{H}.H(k, a), z \leftarrow \text{P}_{OR}((crs_L, crs_{DH}), (x, T_{DH}), w_{DH}, a, c; R_{OR})$ 
     $\pi \leftarrow (a_{NDH}, z_{NDH}, T_{DH}, a, z)$ 
  else  $\pi \leftarrow \emptyset$ 
   $(State_A, cont, d) \leftarrow \mathcal{A}(1^\lambda, State_A, \pi)$ 
until  $cont = \text{false}$ 
return  $d = 0$ 

```

Then we have the following reductions:

- $\mathcal{H}_0, \mathcal{H}_1$: Assuming there exists a PPT algorithm \mathcal{A} that $\left| \Pr[out^{\mathcal{H}_0}] - \Pr[out^{\mathcal{H}_1}] \right| \geq \delta(\lambda)$, then we can construct an adversary \mathcal{A}' that can break the programmability of CIHF \mathcal{H} (Definition 14) through the following reduction:
 - \mathcal{A}' queries the challenger of the programmability of \mathcal{H} that sends back the hash key k
 - \mathcal{A}' samples crs_L by using Gen_L , samples $crs_{DH}, crs_{NDH}, (g, h)$ correspondingly, and does $crs \leftarrow (crs_L, crs_{DH}, crs_{NDH}, (g, h), k)$
 - \mathcal{A}' sends crs to \mathcal{A} to get (x, w)
 - \mathcal{A}' preparing π by using (x, w) and sends it to \mathcal{A}
 - \mathcal{A}' outputs the output of \mathcal{A}
 We now observe that if the challenger uses $\mathcal{H}.Gen$ to sample k , we are in \mathcal{H}_0 , otherwise, we are in \mathcal{H}_1 . This implies $\mathcal{H}_0 \approx \mathcal{H}_1$.
- $\mathcal{H}_1, \mathcal{H}_2$: Assuming there exists a PPT algorithm \mathcal{A} that $\left| \Pr[out^{\mathcal{H}_1}] - \Pr[out^{\mathcal{H}_2}] \right| \geq \delta(\lambda)$, then we can construct an adversary \mathcal{A}' that can break the SHVZK of Σ_{NDH} through the following reduction:
 - \mathcal{A}' queries the challenger of the SHVZK of Σ_{NDH} that sends back the proof a_{NDH}, z_{NDH}
 - \mathcal{A}' samples crs_L by using Gen_L , samples k by using Samp , samples $crs_{DH}, crs_{NDH}, (g, h)$ correspondingly, and does $crs \leftarrow (crs_L, crs_{DH}, crs_{NDH}, (g, h), k)$
 - \mathcal{A}' sends crs to \mathcal{A} to get (x, w)
 - \mathcal{A}' preparing π_{OR} by using (x, w) and does $\pi \leftarrow (\pi_{OR}, T, a_{NDH}, z_{NDH})$
 - \mathcal{A}' sends π to \mathcal{A} , and outputs the output of \mathcal{A}

We now observe that if the challenger provides a real transcript, we are in \mathcal{H}_1 , otherwise, we are in \mathcal{H}_2 . This implies $\mathcal{H}_1 \approx \mathcal{H}_2$.

- $\mathcal{H}_2, \mathcal{H}_3$: Assuming there exists a PPT algorithm \mathcal{A} that $\left| \Pr[out^{\mathcal{H}_2}] - \Pr[out^{\mathcal{H}_3}] \right| \geq \delta(\lambda)$, then we can construct an adversary \mathcal{A}' that can break the

DDH hardness assumption (Definition 1) through the following reduction:

- \mathcal{A}' queries the challenger of the DDH hardness assumption that sends back the tuple $T = (g, h, X, Y)$
- \mathcal{A}' gets (g, h) from T , and uses T to generate $(a_{\text{NDH}}, z_{\text{NDH}})$ from using Sim_{NDH}
- \mathcal{A}' samples crs_L by using Gen_L , samples k by using Samp , samples $crs_{\text{SDH}}, crs_{\text{NDH}}$ correspondingly, and does $crs \leftarrow (crs_L, crs_{\text{SDH}}, crs_{\text{NDH}}, (g, h), k)$
- \mathcal{A}' sends crs to \mathcal{A} to get (x, w)
- \mathcal{A}' preparing π_{OR} by using (x, w) and does $\pi \leftarrow (\pi_{\text{OR}}, T, a_{\text{NDH}}, z_{\text{NDH}})$
- \mathcal{A}' sends π to \mathcal{A} , and outputs the output of \mathcal{A}

We now observe that if the challenger provides a non-DH tuple, we are in \mathcal{H}_2 , otherwise, we are in \mathcal{H}_3 . This implies $\mathcal{H}_2 \approx \mathcal{H}_3$.

- $\mathcal{H}_3, \mathcal{H}_4$: Assuming there exists a PPT algorithm \mathcal{A} that $\left| \Pr[out^{\mathcal{H}_3}] - \Pr[out^{\mathcal{H}_4}] \right| \geq \delta(\lambda)$, then we can construct an adversary \mathcal{A}' that can break the WI of Π_{OR} through the following reduction:

- \mathcal{A}' queries the challenger of the WI of Π_{OR} that sends back $x, \pi_{\text{OR}} = (a_{\text{OR}}, z_{\text{OR}})$
- \mathcal{A}' samples crs_L by using Gen_L , samples k by using Samp , samples $crs_{\text{SDH}}, crs_{\text{NDH}}, (g, h)$ correspondingly, and does $crs \leftarrow (crs_L, crs_{\text{SDH}}, crs_{\text{NDH}}, (g, h), k)$
- \mathcal{A}' sends crs to \mathcal{A} to get (x, w)
- \mathcal{A}' use π_{OR} from the challenger, and does $\pi \leftarrow (\pi_{\text{OR}}, T_{\text{DH}}, a_{\text{NDH}}, z_{\text{NDH}})$
- \mathcal{A}' sends π to \mathcal{A} , and outputs the output of \mathcal{A}

We now observe that if the challenger provide Π_{OR} by using w , where $(x, w) \in R_L$, we are in \mathcal{H}_3 , otherwise we are in \mathcal{H}_4 . This implies $\mathcal{H}_3 \approx \mathcal{H}_4$.

We can concludes $\mathcal{H}_0 \approx \mathcal{H}_1 \approx \mathcal{H}_2 \approx \mathcal{H}_3 \approx \mathcal{H}_4$. Therefore, the simulated transcript from Sim is computationally indistinguishable from a transcript in the real game. \square

On the Adaptive Soundness of Our Protocol. In the previous section, we showed that Π is (non-adaptive) sound and adaptive multi-theorem ZK. In this section, we argue that it is possible to slightly modify Π and get a protocol that enjoys the same properties as Π , but in addition, it is also adaptive-sound.

In [9] the authors show that if the input of the hash function used in the FS transform contains also the theorem (and not just the first round of the underlying protocol), and moreover the trapdoor sigma-protocol is *instance-independent* then the resulting NIZK is adaptive sound. As an additional contribution, the

authors of [9] show that any sigma-protocol can be turned into an instance-independent trapdoor sigma-protocol (this construction has an overhead, that requires computing two ciphertexts for each bit of the challenge of the starting trapdoor sigma-protocol).

Hence, using the results of [9], we can construct an instance-independent trapdoor sigma-protocol for the language $L \vee L_{DH}$. If we apply the FS transform using as the input of the hash-function also x then the final NIZK protocol we obtain is both adaptive sound and adaptive multi-theorem ZK.

A SID Trapdoor Sigma-Protocols for Non-DH Tuples and Proofs

In this section, we present a new sigma-protocol (denoted as Σ_{NDH}) for $L_{NDH} = \{(g, h, X, Y) \mid (g, h) \in S_1 \wedge (X, Y) \in S_2 \wedge \phi(g, h, X, Y) = 1\}$, where $S_1 = \{(g, g^x) \in G \times G \mid x \in \mathbb{Z}_p\}$, $S_2 = \{(h, h^y) \in G \times G \mid y \in \mathbb{Z}_p\}$, and $\phi(g, h, X, Y) = 1$ if and only if $\exists w, w' \in \mathbb{Z}_p : X = g^w \wedge Y = h^{w'} \wedge w \neq w'$. We also argue that the application of the FS transforms on our protocol yields a semi-adaptive multi-theorem sound NIZK.

We propose the formal description of our protocol $\Sigma_{NDH} = (\text{Gen}_{NDH}, \text{P}_{NDH}, \text{V}_{NDH})$ in Fig. 3, where the $crs = \emptyset$.

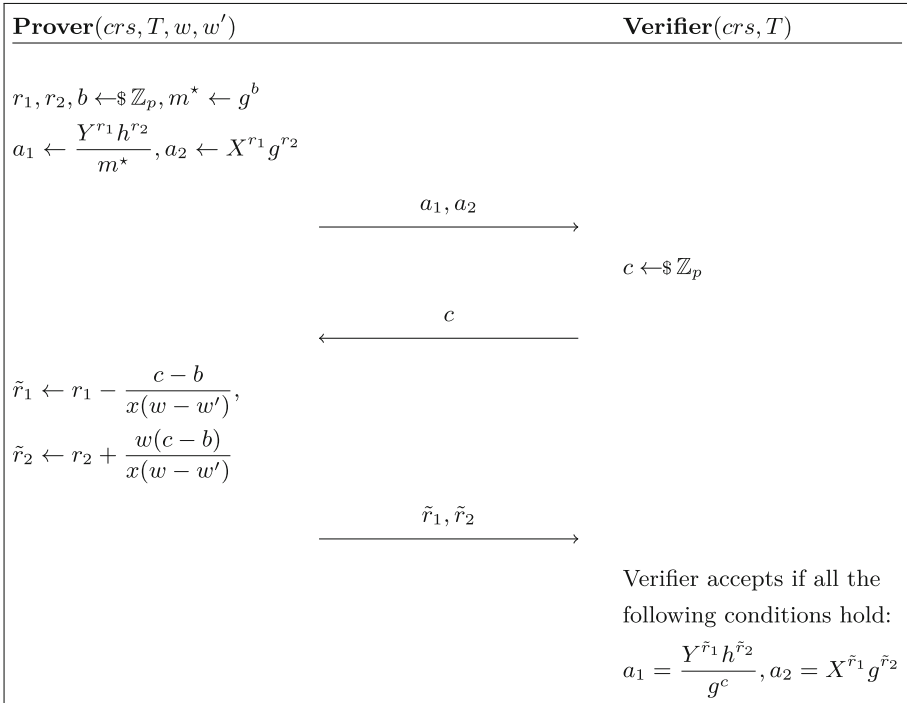


Fig. 3. The protocol for L_{NDH}

Lemma 7. Σ_{NDH} is a sigma-protocol for language L_{NDH} .

Due to lack of space, the proof for Lemma 7 appears in the full version.

Lemma 8. Σ_{NDH} has a PPT extractor $\text{Ext}_{\text{uni}}(\alpha, \tau, a)$, where α is (g, h) from the tuple $T = (g, h, X, Y)$, τ is the trapdoor, a is the first round message, s.t. $\forall T \notin L_{\text{NDH}}$, if the unique bad-challenge is c , Ext_{uni} can extract g^c (which is also unique).

Proof. Ext_{uni} , on input $\alpha = (g, h)$, $\tau = x$, such that $g^x = h$, $a = (a_1, a_2)$ (the first round of the sigma-protocol Σ_{NDH}), returns $g^c \leftarrow \frac{a_2^\tau}{a_1}$, where c is the bad-challenge.

Ext_{uni} outputs the correct results due to the following observation. If we have the first round message $a = (a_1, a_2)$, and $T \notin L_{\text{NDH}}$, due to the optimal soundness property, we know that there is at most one challenge c that makes the transcript (a, c, z) accepting. Then because the transcript is accepting it must be that $a_1 = \frac{Y^{\tilde{r}_1} h^{\tilde{r}_2}}{g^c}$ and $a_2 = X^{\tilde{r}_1} g^{\tilde{r}_2}$. When $T \notin L_{\text{NDH}}$, $\frac{a_2^\tau}{a_1} = \frac{X^{x\tilde{r}_1} g^{x\tilde{r}_2}}{Y^{\tilde{r}_1} h^{\tilde{r}_2}} g^c = g^c$. Because g is the generator in the cyclic group G , g^c and c are 1 to 1 mapping. It means g^c is also unique.

Claim. If the number of all the possible challenges c is bounded to $\text{poly}(\lambda)$, then by using brute force, computing c from g^c is efficient (polynomial time in λ).

Lemma 9. If the challenge c of the protocol Σ_{NDH} satisfies that $c \in \{0, 1\}^{\mathcal{K} \log_2(\lambda^\epsilon)}$ for $\epsilon > 0$ and for integer $\mathcal{K} \geq 1$, then for $t = \Omega(\frac{\lambda^\epsilon}{\mathcal{K} \log_2(\lambda^\epsilon)})$, the parallel repetition version Σ_{NDH}^t is a semi-instance-dependant trapdoor sigma-protocol, for $L_{\text{NDH}} = \{(g, h, X, Y) \mid (g, h) \in S_1 \wedge (X, Y) \in S_2 \wedge \phi(g, h, X, Y) = 1\}$, where $S_1 = \{(g, g^x) \in G \times G \mid x \in \mathbb{Z}_p\}$, $S_2 = \{(h, h^y) \in G \times G \mid y \in \mathbb{Z}_p\}$, and $\phi(g, h, X, Y) = 1$ if and only if $\exists w, w' \in \mathbb{Z}_p : X = g^w \wedge Y = h^{w'} \wedge w \neq w'$.

Proof. By applying Lemma 1 in [12], we know Σ_{NDH}^t is a sigma-protocol, and we only focus on proving the property for the SID trapdoor sigma-protocol.

The corresponding $\text{TrapGen}_{\text{NDH}}$ algorithm has the following inputs:

- 1^λ : The unary representation of the security parameter
- α : (g, h) from the tuple (g, h, X, Y)
- aux : x from $g^x = h$

Then the outputs of $\text{TrapGen}_{\text{NDH}}$ is $crs = \emptyset$ and $\tau = aux$.

We denote the transcript of i -th repetition as (a_1^i, a_2^i, c_i, z_i) . The construction of the bad-challenge extractor $\text{BadChallenge}_{\text{NDH}}(\tau, crs, \alpha, a)$ is:

```

BadChallengeNDH( $\tau, crs, \alpha, a$ ) :
   $g^{c_1} \leftarrow \text{Ext}_{\text{uni}}(\alpha, \tau, (a_1^1, a_2^1))$ 
  Brute force search on  $g^{c_1}$  to get  $c_1$ 
  ...
   $g^{c_t} \leftarrow \text{Ext}_{\text{uni}}(\alpha, \tau, (a_1^t, a_2^t))$ 
  Brute force search on  $g^{c_t}$  to get  $c_t$ 
   $c \leftarrow (c_1 || c_2 || \dots || c_t)$ 
  return  $c$ 
    
```

where $a = ((a_1^1, a_2^1), \dots, (a_1^t, a_2^t))$ is the first round message of Σ_{NDH}^t . By Lemma 8 and the claim that brute force is efficient for small search space, we know $\text{BadChallenge}_{\text{NDH}}$ is a PPT algorithm.

Then we prove the CRS Indistinguishability and Correctness:

- CRS Indistinguishability:
 - Because the Σ_{NDH}^t 's CRS is an empty set, the honestly generated CRS is computationally indistinguishable from CRS computed by $\text{TrapGen}(1^\lambda, x, aux)$
- Correctness: Assuming the Correctness does not hold, it means the transcript of one of the repetitions is not accepted. It contradicts Lemma 8.

□

A.1 Efficiency analysis

Here we compare the efficiency of our Π_{NDH} with the NIZK protocol obtained by applying the FS transform using a CIHF to the well-known protocol $\Sigma_{\text{DH}} = (\text{Gen}_{\text{DH}}, \text{P}_{\text{DH}}, \text{V}_{\text{DH}})$ used to prove that a tuple is non-DH tuple. We recall how such a protocol works in Fig. 4.

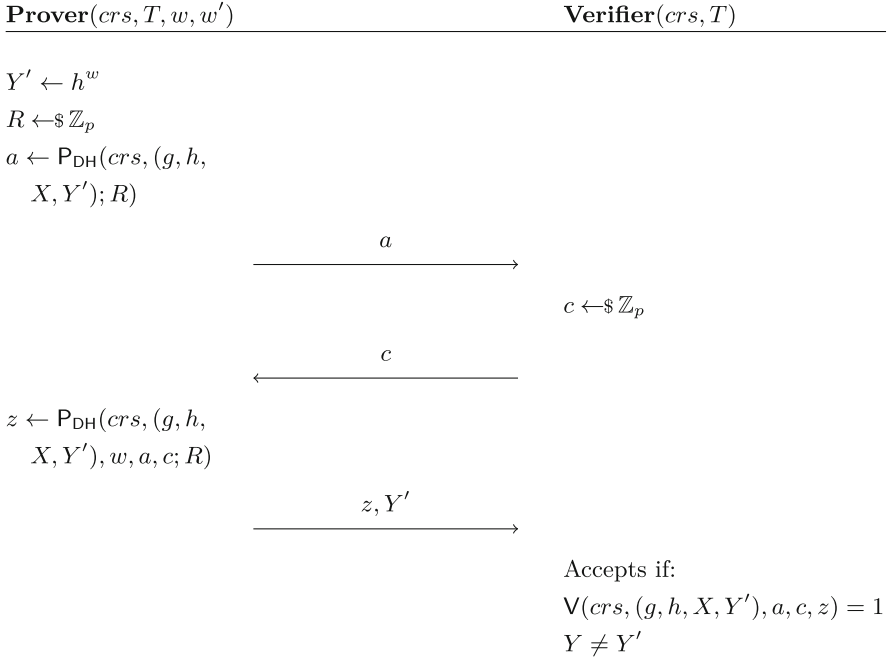


Fig. 4. The protocol for non-DH from DH

Because no expensive operations are introduced in this conversion, the efficiency is the same as Σ_{DH} . Also, the FS transform does not introduce any expensive operations.

Hense, we compare the efficiency of Σ_{NDH}^t with Σ_{DH}^t from Theorem 2:

- Considering the security parameter $\lambda = 2048$, and $\epsilon = \frac{10}{11}$. Then p is 1024 bits.
- Π_{DH} : It requires 1024 repetitions, and in each repetition, the prover needs to compute 2 exponentiations, and the verifier needs to compute 4 exponentiations. In total, it requires 2048 exponentiations for the prover and 4096 exponentiations for the verifier.
- Π_{NDH} : It requires $\frac{1024}{\mathcal{K} \log_2(1024)} = \frac{103}{\mathcal{K}}$ repetitions.
 - If we make $\mathcal{K} = 10$, then the required repetition is 11. In each repetition, the prover needs to compute 5 exponentiations and the verifier needs to compute 5 exponentiations. In total, the prover needs to compute 55 exponentiations and the verifier needs to compute 55 exponentiations.
 - We also want to emphasize that, reducing the number of repetition only influence the reduction of soundness. In the honest execution, neither the prover nor the verifier does the brute force search computation to get c from g^c .

Also, we can use following formula to get lower bound of λ , s.t. Π_{NDH} more efficient than Π_{DH} :

$$6\lambda^\epsilon \geq 10 \frac{\lambda^\epsilon}{\mathcal{K}\epsilon \log_2(\lambda)}$$

$$\log_2(\lambda) \geq \frac{10}{6\mathcal{K}\epsilon}$$

$$\lambda \geq 2^{\frac{5}{3\mathcal{K}\epsilon}}$$

References

1. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93, pp. 62–73. ACM Press (1993). <https://doi.org/10.1145/168588.168596>
2. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: 20th ACM STOC, pp. 103–112. ACM Press (May 1988). <https://doi.org/10.1145/62212.62222>
3. Brakerski, Z., Koppula, V., Mour, T.: NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 738–767. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_26
4. Canetti, R., et al.: Fiat-Shamir: from practice to theory. In: Charikar, M., Cohen, E. (eds.) 51st ACM STOC, pp. 1082–1090. ACM Press (2019). <https://doi.org/10.1145/3313276.3316380>
5. Canetti, R., Chen, Y., Reyzin, L.: On the correlation intractability of obfuscated pseudorandom functions. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 389–415. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_17
6. Canetti, R., Chen, Y., Reyzin, L., Rothblum, R.D.: Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 91–122. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_4
7. Canetti, R., Sarkar, P., Wang, X.: Triply adaptive UC NIZK. Cryptology ePrint Archive, Report 2020/1212 (2020). <https://eprint.iacr.org/2020/1212>
8. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_7
9. Ciampi, M., Parisella, R., Venturi, D.: On adaptive security of delayed-input sigma protocols and Fiat-Shamir NIZKs. In: Galdi, C., Kolesnikov, V. (eds.) SCN 2020. LNCS, vol. 12238, pp. 670–690. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57990-6_33
10. Ciampi, M., Persiano, G., Siniscalchi, L., Visconti, I.: A transform for NIZK almost as efficient and general as the fiat-Shamir transform without programmable random oracles. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 83–111. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0_4
11. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_19

12. Damgård, I.: On Σ -protocol. <http://www.cs.au.dk/~ivan/Sigma.pdf> (2010)
13. De Santis, A., Micali, S., Persiano, G.: Non-interactive zero-knowledge proof systems. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 52–72. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_5
14. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the fiat-Shamir transform. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34931-7_5
15. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In: 31st FOCS, pp. 308–317. IEEE Computer Society Press (1990). <https://doi.org/10.1109/FSCS.1990.89549>
16. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
17. Fischlin, M., Rohrbach, F.: Single-to-multi-theorem transformations for non-interactive statistical zero-knowledge. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12711, pp. 205–234. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_8
18. Holmgren, J., Lombardi, A.: Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications). In: Thorup, M. (ed.) 59th FOCS, pp. 850–858. IEEE Computer Society Press (2018). <https://doi.org/10.1109/FOCS.2018.00085>
19. Kalai, Y.T., Rothblum, G.N., Rothblum, R.D.: From obfuscation to the security of fiat-Shamir for proofs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 224–251. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_8
20. Lapidot, D., Shamir, A.: Publicly verifiable non-interactive zero-knowledge proofs. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 353–365. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-38424-3_26
21. Libert, B., Nguyen, K., Peters, T., Yung, M.: One-shot fiat-Shamir-based NIZK arguments of composite residuosity and logarithmic-size ring signatures in the standard model. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022. EUROCRYPT 2022. LNCS, vol. 13276, pp. 488–519. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-07085-3_17
22. Peikert, C., Shiehian, S.: Noninteractive zero knowledge for np from (plain) learning with errors. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 89–114. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_4



Game-Theoretically Secure Protocols for the Ordinal Random Assignment Problem

T.-H. Hubert Chan^(✉) , Ting Wen, Hao Xie, and Quan Xue

Department of Computer Science, The University of Hong Kong,
Pok Fu Lam, Hong Kong
{hubert,twen}@cs.hku.hk, {hxie,csxuequan}@connect.hku.hk

Abstract. We study game-theoretically secure protocols for the classical ordinal assignment problem (aka matching with one-sided preference), in which each player has a total preference order on items. To achieve the fairness notion of equal treatment of equals, conventionally the randomness necessary to resolve conflicts between players is assumed to be generated by some trusted authority. However, in a distributed setting, the mutually untrusted players are responsible for generating the randomness themselves.

In addition to standard desirable properties such as fairness and Pareto-efficiency, we investigate the game-theoretic notion of maximin security, which guarantees that an honest player following a protocol will not be harmed even if corrupted players deviate from the protocol. Our main contribution is an impossibility result that shows no maximin secure protocol can achieve both fairness and ordinal efficiency. Specifically, this implies that the well-known probabilistic serial (PS) mechanism by Bogomolnaia and Moulin cannot be realized by any maximin secure protocol.

On the other hand, we give a maximin secure protocol that achieves fairness and stability (aka ex-post Pareto-efficiency). Moreover, inspired by the PS mechanism, we show that a variant known as the OnlinePSVar (varying rates) protocol can achieve fairness, stability and uniform dominance, which means that an honest player is guaranteed to receive an item distribution that is at least as good as a uniformly random item. In some sense, this is the best one can hope for in the case when all players have the same preference order.

Keywords: Ordinal assignment problem · Distributed protocols · Game-theoretic security

1 Introduction

In this paper, we study secure distributed protocols for the classical *ordinal assignment problem* [4, 9, 10] (aka *matching with one-sided preference*), in which

This work was partially supported by the Hong Kong RGC grants 17203122, 17202121 and 17201220.

there are players \mathcal{N} and items \mathcal{M} , where each player has some total preference order on the items. For ease of illustration, we focus on the case $n = |\mathcal{N}| = |\mathcal{M}|$, even though our results can be readily generalized to the case $|\mathcal{N}| \neq |\mathcal{M}|$.

A *mechanism* takes a preference profile of all players' preference orders and returns a (possibly random) assignment of items to players, where an assignment is a matching between \mathcal{M} and \mathcal{N} , i.e., a bipartite graph in $\mathcal{M} \times \mathcal{N}$, where the degree of each node is at most 1.

From a player's perspective, the result of the mechanism is the probability vector representing the distribution of the item that it receives. A player's preference naturally induces a partial preference order on the probability vectors. We assume that a player prefers to receive some item over having no item.

In the economics literature, several mechanism properties have been investigated.

- *Pareto Efficiency*. Intuitively, this means a mechanism attempts to cater to the preference orders of the players.

An assignment is *stable* (aka *ex-post* Pareto-efficient) if there is no subset S of players who would like to exchange items such that everyone in S gets a more preferred item afterwards; a mechanism is stable if it always returns a stable assignment.

A (random) assignment is *ordinally efficient* (aka *ex-ante* Pareto-efficient) if there does not exist another random assignment such that a non-empty subset S of players receive item probability vectors they strictly prefer than before (while those for players not in S do not change). A mechanism is ordinally efficient if for any player preference profile, it returns an ordinal efficient assignment. One could see that ordinal efficiency is a stronger property than stability.

- *Fairness*. This is also known as equal treatment (of equals), meaning that if two players have an identical preference order, then under the mechanism, the two players should receive identical item distributions.

In this paper, we will consider a stronger notion of fairness that places conditions when two players have an identical preference among a subset of their most preferred items.

- *Truthfulness*. This is also known as *strategyproof*, which means a player does not have the incentive to misreport their preference order to a mechanism.

Distributed Randomness to Achieve Equal Treatment. Observe that randomness is necessary for a mechanism to achieve equal treatment. Typically, in the economics literature [4], one assumes that some trusted central authority will be responsible for generating the randomness in a mechanism, and it suffices to analyze a mechanism as a function that takes a preference profile and returns a distribution of assignments. In contrast, in a distributed setting such as blockchain applications [14], there is no trusted authority and any randomness is generated in a distributed fashion among the players, each of whom may want to receive a more preferable item distribution or behave maliciously to harm other players. Hence, we will explore various security notions for distributed protocols.

Model of Distributed Protocols. We consider the following assumptions that are commonly adopted in blockchain applications: (i) the protocol is distributed and involves only the players (with no trusted authority), (ii) each message can be seen by everyone. Specifically, we consider a synchronized communication model, in which each player can post messages to some broadcast channel (such as a *ledger* [2]). In each round, each agent reads posted messages on the channel from previous rounds, performs some local computation (possibly based on locally generated randomness) and posts new messages to the channel. At the end of the protocol, some publicly agreed deterministic function is applied to the whole transcript of messages to identify the output.

An *honest* agent follows the procedure as specified by the protocol. To distinguish between truthfulness and honesty, we assume that the preference profile is either publicly known or each player has already declared some preference order before the protocol begins.

In this paper, we also distinguish between mechanism and protocol in the following sense. We say that a protocol *realizes* a mechanism, if under honest execution by all players, the protocol produces a random assignment that has the same distribution as specified by the mechanism. When we say that a protocol has a certain mechanism property (such as equal treatment or ordinal efficiency), we mean that the property is satisfied when all players behave honestly in the protocol.

On the contrary, an *adversary* controls some *corrupted* players that may deviate from the protocol. A *Byzantine* adversary can cause a corrupted player to behave arbitrarily, while a *fail-stop* adversary can only cause a corrupted player to abort (i.e., stop sending messages) in a protocol.

Security Notions of Distributed Protocols. The strictest notion of security for a protocol solving the problem is that under any strategy of the adversary, the output of the protocol still has the same distribution as one under honest execution. However, this is impossible even for the simple case of the *fair coin toss* problem [6, 7], in which 2 players wish to agree on a uniformly random bit in $\{0, 1\}$ (with zero bias). To see that this is a special case of the assignment problem, consider 2 players that have the same preference order on 2 items. Then, any stable mechanism that achieves equal treatment is equivalent to returning one of the 2 possible assignments with equal probability. Specifically, Cleve’s impossibility result [7] states that given any protocol for the fair coin toss problem among two players that terminates within a bounded number of rounds, at least one of the players can cause the output to have a non-zero probability bias towards either 0 or 1 by aborting at some point during the protocol. This impossibility result holds even if one assumes ideal cryptographic primitives such as one-way functions.

Game Theoretic Notions of Security. Observe that Cleve’s aforementioned impossibility result states that a player can bias the outcome of the protocol, but not necessarily towards a more favorable one to itself. As opposed to the fair coin problem (in which the goal of the adversary is to introduce bias), in the *lottery* problem [13] (aka *leader election problem*), exactly one of the n players is

chosen as the *winner*. For the lottery problem, a protocol is *maximin* secure [5] if an honest player’s winning probability does not decrease under the strategy of an adversary.

The lottery problem can be solved by an elegant distributed protocol with the help of a non-malleable commitment scheme (e.g., one based on one-way functions [11]). Intuitively, such a scheme allows an agent i to hide some input x_i in a commitment C_i , which behaves like a blackbox to others; later, the agent can decide to open C_i to reveal x_i , but the scheme prevents C_i from opening to any other different value. To simplify our description, we assume the existence of an ideal commitment scheme; this has the advantage of separating the computational issue regarding cryptography from the game theoretic aspects of the problem.

The special case of $n = 2$ agents can be solved by a simple Blum duel protocol [3], in which each of two players (labeled 0 and 1) randomly picks an input bit in $\{0, 1\}$ and broadcasts its commitment. After receiving another player’s commitment, each player opens its own commitment to reveal its input bit, and the winner is the XOR of the two revealed input bits. However, if one player does not open its input bit, then the other player will automatically be the winner of the duel. Observe that an honest player wins with a probability of at least $\frac{1}{2}$. (Since this is a zero sum game, no dishonest player can win with a probability of larger than $\frac{1}{2}$.) Using a binary tournament tree of depth $O(\log n)$ in which every internal node corresponds to an instance of the duel subroutine, one can see that this readily corresponds to a distributed protocol with $O(\log n)$ rounds for the lottery problem in which an honest player wins with a probability of at least $\frac{1}{n}$.

Motivated by the lottery problem, the notion of maximin security can also be applied to a protocol for the assignment problem, in which an honest player would receive the same or a more preferable outcome distribution, should corrupted players deviate from the protocol.

1.1 Technical Challenges

To understand this game-theoretic notion of security, we first investigate whether well-known mechanisms in the literature can be realized by maximin secure protocols.

Random Priority (RP aka *random serial dictatorship*) mechanism [1, 17]. The mechanism first samples a uniformly random permutation on the players, who are assigned items sequentially, one player at a time accordingly. When it is a player’s turn, it will receive its most preferred item among the still available items. It can be easily checked that the mechanism is truthful and achieves equal treatment, but it is known to be not ordinally efficient.

Observe that to realize RP, one possible approach is to generate a permutation uniformly at random in a “maximin secure” fashion. Since we already have a maximin secure protocol for the lottery problem, it is tempting to use it

to generate a random permutation of players. For instance, an instance of the lottery problem can determine which player ranks first, and so on for the rest of the permutation. Indeed, one can show that this protocol is maximin secure with respect to the rank received by a player in the permutation.

Unfortunately, this does not translate to the maximin security with respect to a player's preference for items. Consider the following example with 3 players such that players 1 and 2 both have item A as their favorite, while player 3 has a different favorite item B . Observe that under honest execution, player 1 receives its favorite item A with probability $\frac{1}{2}$.

However, player 1 can be hurt in the following way. When rank 1 is determined in the first lottery problem instance, player 3 has an abort strategy that it aborts whenever in the round against player 2, which results in player 2 winning automatically in this round. Under this strategy of player 3, player 1 still wins with a probability $\frac{1}{3}$, but the winning probability of player 3 can be transferred to player 2 who now wins with probability $\frac{2}{3}$. As a result, under this attack, player 1 receives its favorite item with probability $\frac{1}{3}$, which is smaller than before.

The above example shows that generating a random permutation via the lottery problem protocol cannot achieve maximin security for the assignment problem, but does not rule out the possibility that there may be a maximin secure protocol that can realize RP.

Probabilistic serial (PS) mechanism. This was proposed by Bogomolnaia and Moulin [4], for which we imagine that each object is one unit of different juice, and each player consumes its most preferred available juice in the order of its preference list at the same rate; the resulting consumption corresponds to a (deterministic) fractional assignment of the object which is a bistochastic matrix¹ that can be random rounded to give an (integral) assignment. While it is known that PS is ordinally efficient and clearly achieves equal treatment, there are known examples in which PS is not truthful. Indeed, it has been proved [4] that in general, no mechanism can simultaneously achieve equal treatment, truthfulness and ordinal efficiency. Realizing PS by a maximin secure protocol seems tricky, because the rounding of the aforementioned bistochastic matrix involves intricate dependencies of item preferences among the players. Indeed, our main result shows that this is actually impossible.

1.2 Our Contributions

Analogous to the aforementioned impossibility result [4] that no mechanism can satisfy equal treatment, truthfulness and ordinal efficiency simultaneously, we have the following impossibility result for maximin secure protocols.

Theorem 1 (Impossibility Result to Achieve Maximin Security). *For $n \geq 4$ players, any mechanism that achieves both strong equal treatment and ordinal efficiency cannot be realized by a maximin secure protocol (against a fail-stop adversary) that terminates with a bounded number of rounds.*

¹ A bistochastic matrix is one with non-negative real elements such that the sum of every row and the sum of every column is equal to 1.

Here are the implications of this impossibility result.

- Strong equal treatment means that for any $k \leq n$, if two players have exactly the same preference order among their k most favorite items, then the two players have exactly the same probabilities for receiving each of those k items. Since PS also achieves strong equal treatment, it follows that no maximin secure protocol can realize PS.
- The impossibility result also means that if a protocol ensures that an honest player will not be hurt by corrupted players (i.e., maximin security is achieved), then the mechanism is not ordinally efficient, which implies that it is possible that all the players might collude and deviate from the protocol such that no player will get hurt and some player will be strictly better off.

Even though we do not know how to realize RP with a maximin secure protocol and have shown that PS cannot be realized by a maximin secure protocol (against even a fail-stop adversary), we have the following positive result on maximin secure protocols.

Theorem 2 (Maximin Secure Protocol). *There exists a mechanism that achieves both stability and strong equal treatment (when all players are honest) and can be realized by a maximin secure protocol against a fail-stop adversary controlling up to $n - 1$ corrupted players.*

Our protocol known as *preference priority* (PP, Algorithm 1) runs a sequence of lottery problem instances, where each lottery decides the fate of a specific item. Loosely speaking, the protocol can achieve maximin security because it ensures that a fail-stop adversary cannot affect the order of the lottery problem instances in the sequence. In Sect. 2, we explain some scenarios in which it is justifiable to consider only fail-stop adversaries.

Uniform Dominance. The notion of maximin security guarantees that an honest player cannot be hurt by corrupted players that deviate from the protocol, but an honest player can still be attacked if other players lie about their preference orders. If a mechanism satisfies equal treatment, then an honest player can be attacked by a malicious adversary that controls every other player, who claims to have exactly the same preference order as the honest player, thereby forcing everyone to receive every item with the same probability. Therefore, the adversary can make sure that an honest player cannot get something better than a uniformly random item in \mathcal{M} . We say that a protocol achieves *uniform dominance* if an honest player receives an item distribution that is at least as good as a uniformly random item, no matter what the other players say (about their preference orders) or do (in the protocol).

It is not too difficult to check that the above idea of realizing RP by generating a uniformly random permutation via instances of the lottery problem can achieve uniform dominance. On the other hand, the PS mechanism ensures that for any preference profile, each player receives an item distribution that is at least as good as a uniformly random item. Even though we have shown that PS cannot be realized by a maximin secure protocol, we have designed a variant known as *online PS with varying rates* (OnlinePSvar, Algorithm 2) that achieves uniform dominance.

Theorem 3 (Uniform Dominance). *The OnlinePSVar protocol achieves stability, strong equal treatment and uniform dominance against a Byzantine adversary (controlling up to $n - 1$ players).*

Even though OnlinePSVar also uses the lottery problem subroutine, it might have a potential advantage over RP when players have vastly different preference orders. For RP, observe that all players need to participate in the lottery problem to determine which player has ranked 1 in the permutation. On the other hand, one can check that for OnlinePSVar, when players have very different favorite items, each instance of the lottery problem can potentially involve fewer players (because each item might be fractionally consumed by fewer players), thereby improving the round complexity of the protocol, as the lottery problem on n players takes $O(\log n)$ rounds.

Paper Organization. We give the formal notation in Sect. 2 and introduce standard building blocks in Sect. 3. Our impossibility result in Theorem 1 is proved in Sect. 4. Our maximin secure protocol is given in Sect. 5, and we show how uniform dominance is achieved in Sect. 6. Finally, we outline some future directions in Sect. 7.

1.3 Other Related Work

Since the ordinal assignment problem was introduced by Gardenfors [9], there have been numerous works on the subject; for details, refer to Chap. 2 of the book [8].

To circumvent Cleve’s aforementioned impossibility result [7] for the fair coin toss problem (with multi-players), Chung et al. [6] have proposed game-theoretic notions of security when players have a preference for the coin outcome.

The folklore tournament tree protocol for the lottery problem has gained renewed interest in the context of blockchain applications [13]. To improve the round complexity of lottery protocols, Chung et al. [5] have considered approximate game-theoretic notions of security. Since the lottery problem has a clear zero-sum game structure, an honest player cannot be hurt *iff* corrupted players cannot gain any unfair advantage. In contrast, our impossibility result in Theorem 1 for the ordinal assignment problem implies that if an honest player cannot be hurt in a protocol, then it might still be possible for some players to collude and be strictly better off.

2 Preliminaries

Let \mathcal{N} denote the set of players and \mathcal{M} denote the set of items, where $n = |\mathcal{N}| = |\mathcal{M}|$. For a positive integer ℓ , we write $[\ell] := \{1, 2, \dots, \ell\}$. We use $\mathcal{S}_{\mathcal{M}}$ (or \mathcal{S} when \mathcal{M} is clear from context) to denote the collection of total orders over \mathcal{M} . Each player $i \in \mathcal{N}$ has some preference order \succ_i in \mathcal{S} , which is also represented by a *favorite* function $O_i : [n] \rightarrow \mathcal{M}$, where $O_i(k)$ is the k -th favorite item of player i .

Given a preference profile $\sigma \in \mathcal{S}^{\mathcal{N}}$, we implicitly assume that the associated \succ_i (also denoted as σ_i) and O_i are defined for each $i \in \mathcal{N}$.

We use \mathcal{A} to denote the collection of *assignment* matrices in $\{0, 1\}^{\mathcal{N} \times \mathcal{M}}$ such that every row and column has exactly one non-zero entry. For instance, given some $P \in \mathcal{A}$, $P(i, j) = 1$ iff player i receives item j ; we also use P_i to denote the i -th row of P . The convex hull² $\text{conv}(\mathcal{A}) = \{P \in [0, 1]^{\mathcal{N} \times \mathcal{M}} : \sum_{j \in \mathcal{M}} P(i', j) = 1, \sum_{i \in \mathcal{N}} P(i, j') = 1, \forall i' \in \mathcal{N}, j' \in \mathcal{M}\}$ is exactly the collection of bistochastic matrices.

Distribution. Given some set \mathcal{U} , we use $\Delta(\mathcal{U}) := \{x \in [0, 1]^{\mathcal{U}} : \sum_{u \in \mathcal{U}} x_u = 1\}$ to denote the collection of distributions on \mathcal{U} .

Mechanism. In this paper, a *mechanism* is a mapping that takes a preference profile in $\mathcal{S}^{\mathcal{N}}$ and returns a distribution in $\Delta(\mathcal{A})$. Typically, the description of a mechanism gives a method to randomly sample an assignment in \mathcal{A} . Observe that a distribution $\rho \in \Delta(\mathcal{A})$ induces a bistochastic matrix $\sum_{A \in \mathcal{A}} \rho_A \cdot A \in \text{conv}(\mathcal{A})$. Alternatively, in the literature, a mechanism is sometimes described by giving the resulting bistochastic matrix, from which a (possibly non-unique) distribution of assignments can be computed efficiently. However, note that it can be NP-hard to compute the bistochastic matrix from a mechanism description (such as RP [16]).

Conventionally, the randomness used for sampling an assignment in a mechanism is assumed to be generated by some trusted authority. *Truthfulness* refers to whether a player reveals its true preference order to the mechanism. The main focus of this work is the scenario when this randomness is jointly generated by the players according to some procedure known as a (distributed) *protocol*.

Communication Model of Protocols. Players participate in a (possibly randomized) protocol, at the end of which the whole transcript of all sent messages determines an assignment in \mathcal{A} . We assume that either the preference profile is public information, or before the protocol begins, each player declares its preference order. We emphasize the distinction that *honesty* refers to whether a player follows the procedure as specified by the protocol, as opposed to whether a player is truthful about its preference.

A protocol proceeds in synchronous rounds over a broadcast channel, i.e., a message sent by a player in one round will reach all players at the beginning of the next round. In every round, based on messages received in previous rounds, a player generates randomness and performs local computation as specified by the protocol to generate a message to be sent in this round.

Adversarial Model. An adversary Adv controls some *corrupted* players. The adversary can observe the internal states of the corrupted players and control their actions. We assume that the adversary is *rushing*, i.e., it can wait for the messages from all honest players in a round before it decides the actions of the corrupted players in that round. A *fail-stop* adversary can instruct a

² A convex hull of $S \subset \mathbb{R}^n$ refers to the minimum convex set that contains S .

corrupted player to deviate from the protocol only by stopping to broadcast a message in some round (after which the player will not broadcast any message in subsequent rounds). A Byzantine adversary can instruct a corrupted player to behave arbitrarily. An *adaptive* adversary can decide which so far honest player to corrupt at the end of a round, based on messages already sent. However, since we will mainly consider protocols that are secure against $n - 1$ corrupted players, adaptive corruption is not a crucial feature of the adversary.

Ideal Cryptographical or Hardware Assumptions. Under the following scenarios, we can restrict our attention to fail-stop adversaries.

- *Ideal Cryptographical Assumption.* We consider adversaries that cannot break cryptographical primitives such as commitment schemes [11] and zero-knowledge proofs [15]. At the beginning of the protocol, each player generates all the randomness that will be used in each round of the protocol using *verifiable random functions* [12] and broadcast the commitments of the randomness, together with the corresponding zero-knowledge proofs that the randomness and commitments are generated correctly. Then, in each round of the protocol, a player uses committed randomness to generate and broadcast the message, together with the zero-knowledge proof that the message is generated using the committed randomness.

We remark that for Byzantine adversaries in Theorem 3, we assume only the existence of ideal commitment schemes (but not necessarily zero-knowledge proofs or verifiable random functions).

- *Ideal Hardware Assumption.* Each player is assumed to reside within an SGX enclave that cannot be corrupted. Hence, an adversary can only disrupt the broadcast channel of a player.

We say that a protocol realizes a mechanism if, under honest execution by all players, the protocol produces the same distribution of assignments as the mechanism.

Player Satisfaction. Recall that a mechanism returns some (random) $A \in \mathcal{A}$, where each A_i , the i -th row of A , is a random vector and the j -th element of expectation $E[A_i]$ illustrates the probability that player i gets item j under this (random) mechanism. We could easily see that $E[A_i] \in \Delta(\mathcal{M})$, where $\Delta(\mathcal{M})$ is the collection of distributions of items. Rather than using a utility function that could give a total order for comparison, we say the satisfaction of player i to be $E[A_i]$ which introduces a partial order under what we call vector dominance.

Definition 1 (Vector Dominance). *Given vectors $p, q \in \Delta(\mathcal{M})$, a player i with favorite function O_i prefers p to q if*

$$\forall j \in [n], \sum_{k \in [j]} p(O_i(k)) \geq \sum_{k \in [j]} q(O_i(k)). \tag{1}$$

In this case, we say that p dominates q (with respect to i), and this defines a partial order $p \succeq_i q$ on $\Delta(\mathcal{M})$. Observe that the partial order can be extended to $[0, 1]^{\mathcal{M}}$ (where the coordinates of a vector do not necessarily sum up to 1) also via (1).

Definition 2 (Matrix Dominance). Given $P, Q \in \text{conv}(\mathcal{A})$ and $\sigma \in \mathcal{S}$, we say that P dominates Q (with respect to σ), if, for all $i \in \mathcal{N}$, the rows of P and Q corresponding to i satisfy $P_i \succeq_i Q_i$; we denote this by $P \succeq_\sigma Q$.

Strict Dominance. Observe that we use the term “dominate” to refer to a binary relation \succeq that happens to be reflexive; hence, every element dominates itself. When we say p strictly dominates q , we mean $p \succeq q$ and $p \neq q$.

2.1 Some Well-Known Properties of Mechanisms

The following property intuitively expresses the idea that a mechanism should return an assignment according to the preferences of the players.

Definition 3 (Stability). An assignment $P \in \mathcal{A}$ is stable with respect to a preference profile $\sigma \in \mathcal{S}^{\mathcal{N}}$, if there does not exist a different assignment P' such that $P' \succeq_\sigma P$.

A mechanism is stable if it always produces a stable assignment with respect to the input preference profile.

Definition 4 (Ordinal Efficiency). A bistochastic matrix $P \in \text{conv}(\mathcal{A})$ is ordinally efficient with respect to $\sigma \in \mathcal{S}^{\mathcal{N}}$, if there does not exist a different $P' \in \text{conv}(\mathcal{A})$ such that $P' \succeq_\sigma P$.

A mechanism is ordinally efficient if for all inputs $\sigma \in \mathcal{S}^{\mathcal{N}}$, it returns a distribution in $\Delta(\mathcal{A})$ whose induced bistochastic matrix is ordinally efficient with respect to σ .

Ordinal efficiency is a stronger property than stability. However, if a mechanism returns an assignment based on players’ preferences, then a player may benefit by lying about its true preference.

Definition 5 (Truthfulness). A mechanism is (strongly) truthful, if a player lying about its preference order will receive a vector in $\Delta(\mathcal{M})$ that is dominated (with respect to its true preference) by the vector received had it been truthful.

A mechanism is weakly truthful, if a lying player cannot receive a vector in $\Delta(\mathcal{M})$ that strictly dominates the vector received had it been truthful.

All the properties above can be achieved by a deterministic mechanism (which is realized by a trivial protocol in which no communication other than announcing one’s preference is needed). For instance, in a *deterministic serial dictatorship*, the players can be arbitrarily ranked and we let a higher-ranked player choose its favorite item before lower-ranked players. The following property captures fairness, and can be achieved only with randomness.

Definition 6 ((Strong) Equal Treatment (of Equals)). A matrix $P \in \text{conv}(\mathcal{A})$ achieves (strong) equal treatment with respect to $\sigma \in \mathcal{S}^{\mathcal{N}}$ (that defines favorite functions O_i ’s), if for all $i, j \in \mathcal{N}$ and $\ell \in [n]$, the following holds:

$$“\forall k \in [\ell], O_i(k) = O_j(k)” \text{ implies that } “\forall k \in [\ell], P(i, O_i(k)) = P(j, O_j(k))”.$$

A matrix P achieves weak equal treatment if the above condition holds for $\ell = n$ (but not necessarily for other values of ℓ).

Fact 1 (Impossibility Result [4]). For $n \geq 4$ players, there is no mechanism that can achieve all the following: ordinal efficiency, strong truthfulness and weak equal treatment.

This impossibility result implies that any fair mechanism (in terms of equal treatment) is either (i) not strongly truthful or (ii) not ordinally efficient. In case (i), this means that a player might have the incentive to lie about its preference order. In case (ii), this means that potentially all players might collude and deviate from the protocol such that everyone is better off. Therefore, in this paper, we focus on notions that provide some guarantees to honest and truthful behavior, as opposed to discouraging deceitful or corrupted behavior.

2.2 Security Notions of Protocols

We introduce our security notions for protocols and explain the intuition. The next security notion encourages a player to remain honest even when there are corrupted players, because it captures the guarantee that an honest player will not be hurt.

Definition 7 ((Approximate) Maximin Security). For $\epsilon \geq 0$, a protocol Π is $(1 - \epsilon)$ -maximin secure against an adversary Adv if the following holds for any input preference profile $\sigma \in \mathcal{S}^N$. Given σ , suppose that $Q \in \mathcal{A}$ is the (random) assignment produced by Π under the strategy of Adv , while $P \in \mathcal{A}$ is the one produced had every player behaved honestly. Then, for every honest player i , the expectations of the i -th rows satisfy $\mathbb{E}[Q_i] \succeq_i (1 - \epsilon) \cdot \mathbb{E}[P_i]$, where the partial order \succeq_i is defined in Definition 1 with respect to the preference σ_i of player i .

In this work, we focus on the special case $\epsilon = 0$, which is simply known as maximin secure.

As mentioned in the introduction, if players can lie about their preference orders, then the best guarantee that one can only hope for is that an honest player still receives something that is at least as good as a uniformly random item.

Definition 8 (Uniform Dominance). A protocol Π achieves uniform dominance against an adversary Adv if for any input preference profile $\sigma \in \mathcal{S}^N$ and any honest player i , the i -th row of the (random) assignment $P \in \mathcal{A}$ returned by the protocol (under the strategy of Adv) satisfies $\mathbb{E}[P_i] \succeq_i \mathbf{e}$, where $\mathbf{e} \in \Delta(\mathcal{M})$ is the uniform vector and \succeq_i is the partial order defined in Definition 1 with respect to the preference σ_i of player i .

Remark 1. An equivalent formulation of Definition 8 is that for all $\ell \in [n]$, the probability that a truthful and honest player will receive an item from its top ℓ choices is at least $\frac{\ell}{n}$, no matter what the other players say or do.

3 Standard Building Blocks

We give descriptions for some well-known primitives. Since they are all standard, we just highlight some important properties and give the relevant references.

Commitment Scheme. Assuming the existence of one-way functions/permutations, there is a constant-round publicly verifiable commitment scheme [11] that is perfectly correct, perfectly binding, and concurrent non-malleable. For the purpose of understanding this paper, the reader just needs to know that the *commit phase* of the scheme allows a player to construct a *commitment* C of some secret message m . In a real-world scheme, the commitment is *computationally hiding*, which means a polynomial-time adversary cannot learn anything about the secret message from C . However, for ease of exposition, we will assume that the commitment is ideally secure and the event that the adversary can gain extra information from the commitment has zero probability. In the *open phase*, the player can choose to open the commitment to reveal the secret message m , where perfectly binding means that it is impossible to open the commitment to any other message different from m .

Lottery Problem. There is a set \mathcal{N} of n players, and the input is a probability vector $p \in \Delta(\mathcal{N})$. The goal is for the players to participate in a protocol that determines a winner such that for each $i \in \mathcal{N}$, player i wins with a probability p_i .

Duel Protocol. The special case $n = 2$ for rational input probability vector can be solved by an extension of the Blum's protocol [3] that uses a commitment scheme. On a high level, in the first round, each of the two players picks a random element from some appropriate ring and broadcasts its commitment. In the second round, each player opens its commitment and the sum of the opened elements determines the winner. If a player fails to open its commitment, the other player is the winner; if both players fail to open their commitments, a default player can be the winner. It is straightforward that an honest player i wins with a probability at least p_i even if the other player is controlled by a Byzantine adversary. However, as the duel protocol is used as a subroutine later, there is some subtlety when both players are controlled by the adversary. Observe that a Byzantine adversary can choose which player to be the winner without being detected, while any deviation by a fail-stop adversary will be immediately revealed in the transcript. This distinction is important later as we consider maximin security of protocols.

Tournament Tree Protocol. The duel protocol can be generalized to any $n \geq 2$ players with rational input probability vector by the *tournament tree* protocol that has a binary tree structure in which each internal node corresponds to an instance of a duel protocol; for a detailed description, see [5]. Again, any honest player i wins with a probability at least p_i even if all other players are controlled by a Byzantine adversary. Similarly, as in the duel protocol, if all players are corrupted, a Byzantine adversary can choose any player to be the winner without being detected.

3.1 Augmented Protocols for the Lottery Problem

For completeness, we describe the augmented duel protocol and introduce the terminology to describe the detection of corrupted players, in the case of fail-stop adversaries.

In an instance $\text{AugDuel}(p_1, p_2)$, there are non-negative integers $k_1, k_2 \in \mathbb{Z}$ such that for $i \in \{1, 2\}$, player i is supposed to win with probability $p_i = \frac{k_i}{k_1+k_2}$. In addition to the two players, all players in \mathcal{N} (might) participate as follows.

1. *Commit Step.* Denote $k := k_1 + k_2$ and $\ell := \lceil \log_2 k \rceil$. Each player $i \in \{1, 2\}$ samples a uniformly random element s_i in the ring \mathbb{Z}_k , which can be represented by an ℓ -bit string; each player in $\{1, 2\}$ commits to its string and broadcasts the commitment.
2. *Open Step.* Each player $i \in \{1, 2\}$ opens its commitment to reveal s_i . If $s_1 + s_2 \in \{0, 1, \dots, k_1 - 1\}$, then player 1 wins; else, player 2 wins.
3. *Corruption Detection and Survivor.* If a player in $\{1, 2\}$ aborts or fails to open its commitment to reveal an element in \mathbb{Z}_k , then the protocol identifies this player as *corrupted*. If there is only one identified corrupted player, the other player is the winner; if both players are identified as corrupted, a default winner (say player 1) can be chosen.
A player in $\{1, 2\}$ that neither wins nor is identified as corrupted is known as a *survivor*.

Lemma 1 (Augmented Duel Protocol). *In an instance $\text{AugDuel}(p_1, p_2)$ of the augmented duel protocol, the following properties hold.*

1. *Even when Adv is Byzantine, an honest player $i \in \{1, 2\}$ wins with a probability at least p_i .*
2. *Suppose Adv is fail-stop. Then, there exists a coupling³ between the honest execution and the corrupted execution under the strategy of Adv such that if the survivor sets S and S^{Adv} correspond to the honest and the corrupted executions, respectively, it holds that $S^{\text{Adv}} \subseteq S$; moreover, if an honest player wins in the honest execution, it also wins in the corrupted execution.*

Proof. The first statement for Byzantine adversaries is a well-known result, and we prove the second statement under fail-stop adversaries.

We first describe the coupling. We sample s_1 and s_2 independently from \mathbb{Z}_k and use them to create a coupling between an honest execution and an execution under the strategy of Adv. Observe that $s_0 = s_1 + s_2$ is distributed uniformly at random in \mathbb{Z}_k .

Recall that the goal is to show that by fixing s_1 and s_2 , we always have $S^{\text{Adv}} \subseteq S$, where S is the survivor set under honest execution.

Finally, without loss of generality, assume that conditioning on some value $s_0 = s_1 + s_2$, the survivor set is $S = \{1\}$, which means player 2 is the winner under

³ In probability theory, a coupling between two probability spaces (Ω_1, Pr_1) and (Ω_2, Pr_2) is a joint space $(\Omega_1 \times \Omega_2, \text{Pr})$, whose projections into Ω_1 and Ω_2 equal to (Ω_1, Pr_1) and (Ω_2, Pr_2) , respectively.

honest execution. The only way to contradict $S^{\text{Adv}} \subseteq S$ is to make $2 \in S^{\text{Adv}}$, i.e., 2 cannot be a winner under the strategy of Adv.

Conditioning on this value of s_0 , observe that the only way the adversary Adv can make player 2 lose the duel is to make it fail to open its commitment, thereby identifying player 2 as corrupted; this also means that player 2 cannot lose if it remains honest.

Therefore, it follows that $2 \notin S^{\text{Adv}}$, which means that $S^{\text{Adv}} \subseteq S$; moreover, if 2 is honest, then it also wins in the corrupted execution. \square

Extension to the Tournament Tree Protocol. We can use the augmented AugDuel as a subroutine in the tournament tree protocol. Given a subset $\mathcal{N}' \subseteq \mathcal{N}$, we denote an instance of the augmented tournament tree protocol by AugTourn($p_i : i \in \mathcal{N}'$), where each instance of the duel protocol is implemented by AugDuel. Similarly, a non-winning player of AugTournament that is not identified as corrupted in any AugDuel instance is known as a survivor. A similar result is given as follows.

Lemma 2 (Augmented Tournament Tree Protocol). *In an instance AugDuel($p_i : i \in \mathcal{N}'$) of the augmented tournament tree protocol, the following holds.*

1. *Even when Adv is Byzantine, an honest player $i \in \mathcal{N}'$ wins with a probability at least p_i (even when all other players in \mathcal{N} are corrupted).*
2. *Suppose that Adv is fail-stop. Then, there exists a coupling between an honest execution and the execution under the strategy of Adv with corresponding survivor sets S and S^{Adv} such that it holds that $S^{\text{Adv}} \subseteq S$; moreover, under this coupling, if an honest player wins under the honest execution, it also wins in the corrupted execution.*

Proof. The proof follows from Lemma 1, which gives the first statement.

For the second statement, we apply the same coupling over all instances of AugDuel as in the proof of Lemma 1. Suppose in an honest execution over candidates \widehat{N} , some player i_0 is the winner of AugTourn, which means the survivor set is $S = \widehat{N} \setminus \{i_0\}$. This means that i_0 is the winner of all the AugDuel instances. Under the same conditions as in Lemma 1, there is no way i_0 can lose any of the duels without being identified as a corrupted player. Therefore, $i_0 \notin S^{\text{Adv}}$ and the result follows. \square

4 Impossibility Result to Achieve Maximin Security

The impossibility result in Fact 1 states that in general, no mechanism can achieve strong truthfulness, ordinal efficiency and weak equal treatment simultaneously. Hence, even when all players are honest, no protocol can realize such a mechanism. Recall that we have the distinction between truthfulness (whether a player reveals its true preference) and honesty (whether a player follows a protocol), and the notion of maximin security in Definition 7 is concerned about players' honesty

(as opposed to their truthfulness). Therefore, as a first step to designing protocols, it is natural to ask whether it is possible to have a maximin secure protocol that realizes a mechanism that satisfies ordinal efficiency and strong equal treatment. The goal of this section is the following impossibility result.

Theorem 4 (Impossibility Result). *There exists an instance with $n = 4$ players such that any mechanism that achieves ordinal efficiency and strong equal treatment cannot be realized by a maximin secure protocol (which terminates in a bounded number of rounds) against fail-stop adversaries that control at least $\frac{n}{2}$ players.*

To get some intuition about ordinally efficient mechanisms, we revisit a well-known ordinally efficient mechanism that also achieves strong equal treatment.

Recall that the PS mechanism [4] gives a (deterministic) procedure to compute the induced bistochastic matrix P from a given preference profile. Initially, all items are *unconsumed*. At any moment, each player can fractionally consume its favorite remaining item at a unit rate until that item is totally consumed. Observe that at time 1, all items will be totally consumed. The entry $P(i, j)$ is the fraction of item j consumed by player i in this process.

Example 1 (Instance with 4 Players). Consider $n = 4$ players with the following preference profile σ on $\mathcal{M} := \{m_i : i \in [4]\}$, where the PS mechanism produces the bistochastic matrix $P_{PS} \in \text{conv}(\mathcal{A})$.

$$\begin{array}{l}
 m_1 \succ_1 m_3 \succ_1 m_2 \succ_1 m_4 \\
 m_1 \succ_2 m_4 \succ_2 m_2 \succ_2 m_3 \\
 m_2 \succ_3 m_3 \succ_3 m_1 \succ_3 m_4 \\
 m_2 \succ_4 m_4 \succ_4 m_1 \succ_4 m_3
 \end{array}
 \quad
 P_{PS} = \begin{bmatrix}
 \frac{1}{2} & 0 & \frac{1}{2} & 0 \\
 \frac{1}{2} & 0 & 0 & \frac{1}{2} \\
 0 & \frac{1}{2} & \frac{1}{2} & 0 \\
 0 & \frac{1}{2} & 0 & \frac{1}{2}
 \end{bmatrix}$$

Observe that P_{PS} corresponds to a unique distribution $\Delta(\mathcal{A})$ as follows: $P_{PS} = \frac{1}{2}A_{\text{head}} + \frac{1}{2}A_{\text{tail}}$, where

$$A_{\text{head}} = \begin{bmatrix}
 0 & 0 & 1 & 0 \\
 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 \\
 0 & 0 & 0 & 1
 \end{bmatrix}
 \quad \text{and} \quad
 A_{\text{tail}} = \begin{bmatrix}
 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 \\
 0 & 0 & 1 & 0 \\
 0 & 1 & 0 & 0
 \end{bmatrix}.$$

Lemma 3 (Unique Distribution). *For the problem instance in Example 1, P_{PS} is the unique bistochastic matrix that achieves both ordinal efficiency and strong equal treatment.*

Proof. Suppose $P \in \text{conv}(\mathcal{A})$ is a bistochastic matrix that achieves both ordinal efficiency and strong equal treatment for the instance in Example 1.

First, consider player 1. Because of strong equal treatment, we have $P(1, m_1) = P(2, m_1)$, which implies that $P(1, m_1) \leq \frac{1}{2} = P_{PS}(1, m_1)$. Since $(P_{PS}1, O_1(1)) + P_{PS}(1, O_1(2)) = P_{PS}(1, m_1) + P_{PS}(1, m_3) = 1$, we have

for $\ell \in \{2, 3, 4\}$, $1 = \sum_{j \in [\ell]} P_{\text{PS}}(1, O_1(j)) \geq \sum_{j \in [\ell]} P(1, O_1(j))$.

It follows that the rows of the matrices corresponding to player 1 satisfy: $P_{\text{PS}}(1, \cdot) \succeq_1 P(1, \cdot)$.

A similar analysis for players 2 to 4 implies that $P_{\text{PS}} \succeq_\sigma P$, which means P_{PS} dominates P with respect to P . Since P is ordinally efficient with respect to σ , it follows that $P = P_{\text{PS}}$. □

4.1 Reduction from Coin-Flipping Problem

We next complete the proof of Theorem 4. We assume that there is a maximin secure protocol Π that terminates within a bounded number of rounds, and realizes a mechanism that achieves both equal treatment and ordinal efficiency on the instance in Example 1. From Π , we will construct a two-party coin-flipping protocol. Finally, we show that Cleve’s impossibility result [7] will contradict the maximin security of Π (against fail-stop adversaries controlling at most 2 players).

Interpreting Π as a Two-Party Coin-Flipping Protocol. Suppose party A and part B would like to use Π in Example 1 as a coin-flipping protocol. Party A controls players in $\{1, 2\}$, while party B controls players in $\{3, 4\}$. Observe that we consider the case that at most one party is corrupted by a fail-stop adversary. By Lemma 3, when both parties are honest, the outcome of Π can be either A_{head} or A_{tail} , which can be naturally interpreted as a coin outcome. However, if either party is corrupted, there can be other outcomes of Π that we need to interpret as coin outcomes, after which the description of the coin-flipping protocol will be completed.

The following lemma says that the maximin security of Π implies that when only one party is corrupted, the assignment for players in the honest party still satisfies either assignment matrix A_{head} or A_{tail} . Hence, if possible, we can use the assignment for players in either party A or party B to determine the coin outcome. Note that the assignments for the two parties will not contradict each other. The reason is that combining the first two rows from one of the assignment matrices and the last two rows from the other assignment matrix will not be a valid assignment.

Finally, if the assignment for the players in neither party is consistent with A_{head} or A_{tail} , Lemma 4 implies that both parties are corrupted, in which case any default coin outcome (say head) can be returned.

Lemma 4 (Assignment for an Honest Party). *Suppose Π is maximin secure against a fail-stop adversary controlling at most 2 players. Then, the assignment for players in an honest party in Π agrees with either A_{head} or A_{tail} , each of which happens with probability $\frac{1}{2}$.*

Proof. We consider the case that party $A = \{1, 2\}$ is honest, and the case when party B is honest can be analyzed similarly.

By the maximin security of Π , each player in $\{1, 2\}$ receives m_1 with probability $\frac{1}{2}$ (which means which player receives m_1 partitions the sample space

into two equally likely events). Since the vector received by player 1 must dominate the row $P_{\mathcal{PS}}(1, \cdot)$ (with respect to its own preference), it follows that if player 1 does not receive m_1 (which happens with probability $\frac{1}{2}$), it must receive $m_3 = O_1(2)$.

Similarly, if player 2 does not receive m_1 , then it must receive $m_4 = O_2(2)$. Therefore, it follows that the assignment for players in party A satisfies either A_{head} or A_{tail} , each of which happens with probability $\frac{1}{2}$. \square

Corollary 1 (Contradiction to Cleve’s Result [7]). *Lemma 4 implies that when there is only one corrupted party in the two-party coin flipping protocol, the outcome of the coin is unbiased.*

5 Achieving Perfect Maximin Security

In view of the impossibility results in Fact 1 and Theorem 4, we design a protocol (assuming ideal cryptographic tools) that achieves the following properties.

Theorem 5 (Achieving Stability, Strong Equal Treatment and Maximin Security). *Assuming an ideal commitment scheme, there exists a protocol that realizes a mechanism that achieves stability and strong equal treatment (when all players behave honestly); moreover, the protocol achieves perfect maximin security against a fail-stop adversary that controls up to $n - 1$ players.*

5.1 Preference Priority (PP) Protocol

Algorithm Intuition. The PP protocol in Algorithm 1 proceeds according to round r from 1 to n . In round r , survivors (that have not been assigned their at least $(r - 1)$ -st items) will compete for their r -th favorite items (if still available) via the AugTurn protocol. If the r -th favorite item for a player is no longer available, then the player does not compete for any item in round r . Observe that this implies that the protocol is not weakly truthful (in the case where all players honestly follow the protocol). However, this rigidity of when an item can be assigned is how this protocol achieves maximin security.

Naive Variant. Observe that one could consider a more straightforward variant of Algorithm 1. In each round r , instead of restricting the survivors to compete for their r -th favorite item, we allow them to compete for their most preferred remaining items. Specifically, we replace \mathcal{C} in line 8 in Algorithm 1 by partitioning S according to each survivor’s favorite item in R , i.e., we denote $O_i(R)$ as the favorite item of i in R , and let $\mathcal{C} = \{i \in S : O_i(R) = j\} : j \in R\}$. We call this variant of the algorithm NaivePP.

One can verify that the following Lemma 5 is still valid for NaivePP.

Lemma 5. *The PP protocol described in Algorithm 1 realizes a mechanism that achieves strong equal treatment and stability.*

```

1 Input: A preference profile  $\sigma \in \mathcal{S}^{\mathcal{N}}$  (declared by players) for the items  $\mathcal{M}$ .
2 Output: A (random) assignment  $A \in \mathcal{A}$ .
3 Initialization:
4 Set  $A$  to an empty assignment (i.e., a zero matrix).
5 Let  $S \leftarrow \mathcal{N}$  denote the current collection of valid survivors.
6 Let  $R \leftarrow \mathcal{M}$  denote the current collection of available items.
7 for  $r$  from 1 to  $n$  do
8   Let  $\mathcal{C} = \{\{i \in S : O_i(r) = j\} : j \in R\}$  be a partial partition of  $S$  according
   to each survivor's  $r$ -th favorite item in  $R$ .
9   foreach  $C \in \mathcal{C}$  in an arbitrary order do
10    Suppose for all  $i \in C$ , the common  $r$ -th favorite item is  $O_i(r) = j$ .
11    Let  $\mathbf{p} = (p_i = \frac{1}{|C|} : i \in C)$  indicate that players in  $C$  should compete for
    item  $j$  uniformly at random.
12    Run AugTourn( $\mathbf{p}$ ) to obtain the winner  $\hat{i}$  and the survivor set
     $S' \subseteq C \setminus \{\hat{i}\}$ ; if  $|C| = 1$ , we assume that the only player in  $C$  is the
    default winner and cannot abort.
13    Assign  $A(\hat{i}, j) \leftarrow 1$ .
14    Update  $R \leftarrow R \setminus \{j\}$  and  $S \leftarrow S \setminus (C \setminus S')$ .
15  end
16 end
17 Any remaining items are arbitrarily assigned to players with no items yet
   (according to some pre-determined rule) and update  $A$  accordingly.
18 return assignment  $A$ 

```

Algorithm 1: Preference Priority Protocol

Proof. Strong equal treatment follows because if two players have exactly the same preference for their r most favorite items, then they will behave in exactly the same way as long as those r items are not assigned.

Stability follows because if a player is assigned an item j in the r -th round, then all its more preferred items than j are no longer available at the beginning of the r -th round. \square

However, the following Example 2 shows that NaivePP is not maximin secure.

Example 2 (NaivePP Not Maximin Secure). Consider $n = 9$ players with the preference profile given by the following matrix, where each row corresponds to a player and each entry (i, j) contains the index of the j -th favorite item for player i .

1	2	3	4	5	6	7	8	9
1	2	6	4	5	7	8	9	3
1	2	6	4	5	7	8	9	3
1	2	6	4	5	7	8	9	3
1	2	6	4	5	7	8	9	3
5	2	3	7	8	6	9	1	4
5	2	3	7	8	6	9	1	4
5	3	6	7	8	9	1	2	4
5	3	6	7	8	9	1	2	4

Honest scenario. We first argue that when every player participates honestly, player 1 will surely receive one of its top 4 favorite items. The reason is that player 1 will definitely not get item m_3 , since m_3 will be assigned to one of player 8 or player 9 at round 2. This means if player 1 has not received its top 2 items by the beginning of round 3, it will be the only player to compete for m_4 in round 3.

Corrupted scenario. We show that if both players 8 and player 9 abort in their first round, then with positive probability, player 1 does not get one of its top 4 favorite items.

Observe that with positive probability, player 7 gets m_5 . Then, with positive probability both player 1 and player 6 fail to get m_2 , and hence these two players will compete for m_3 at round 3. With positive probability, player 1 fails to receive its top 3 favorite items, and it has to compete with at least one player (from players 2 to 5) for m_4 at round 4. Hence, we conclude that when player 8 and player 9 abort in their first round, the probability of player 1 getting one of its top 4 favorite items is less than 1.

5.2 Maximin Security Analysis

Simplifying Notation. We first introduce some notations to facilitate the analysis of maximin security. Throughout the analysis, we fix some input preference profile σ and honest player $i_0 \in \mathcal{N}$, and use \succeq to denote the partial order on $\Delta(\mathcal{M})$ defined in Definition 1 with respect to the preference σ_{i_0} of player i_0 .

Execution State. We use Λ to denote the collection of execution states of the protocol in Algorithm 1. A state $\lambda = (r, S, R, a) \in \Lambda$ is a tuple, where the protocol is currently at the beginning of round $r \in [n]$, S is the current collection of survivors, R is the current collection of remaining items, and if $i_0 \notin S$, then $a \in \mathcal{M}$ is the item already received by i_0 , and $a = \perp$ otherwise.

While we consider a state λ that may not be reachable from an honest execution, we only consider *valid* states that satisfy the following conditions:

- $|S| \leq n - r + 1$ and $|S| \leq |R|$;
- if $i_0 \in S$, then all of the $(r - 1)$ most favorite items of i_0 are not in R .

Observe that if i_0 is honest, then only valid states (which are defined with respect to i_0) can be reached in the execution of the protocol.

Item Distribution. For any state $\lambda \in \Lambda$, we use $\Pi(\lambda) \in \Delta(\mathcal{M})$ to denote the distribution of the item received by player i_0 if the protocol is executed honestly by all players from state λ onwards. Since $\Pi(\lambda)$ has no randomness if $i_0 \notin S$ (because i_0 has already received an item from A), it suffices to consider the case $i_0 \in S$ (and $a = \perp$).

Lemma 6 (Monotonicity with Respect to Removing Survivors or Adding Items). *Consider a valid state $\lambda = (r, S, R, \perp) \in \Lambda$, where S contains an honest player i_0 . Then, the following monotone properties hold.*

$P(r)$: Remove Player. Suppose $i \neq i_0$ is another player, and $\lambda_1 = (r, S \setminus \{i\}, R, \perp)$, where we allow $i \notin S$. Then, $\Pi(\lambda_1) \succeq \Pi(\lambda)$.

$Q(r)$: Add Item. Suppose $j \in \mathcal{M} \setminus R$ is an item not in R such that $\lambda_2 = (r, S, R \cup \{j\}, \perp)$ is still valid. Then, $\Pi(\lambda_2) \succeq \Pi(\lambda)$.

Proof. We consider backward induction on r . For the base case $r = n$, because of the definition of a valid state, we have the trivial case that S contains a single survivor i_0 and R contains a single item $O_{i_0}(n)$. Hence, $\lambda = \lambda_1$, and the statement $P(n)$ holds. Observe that there is no other item $j \notin R$ such that $\lambda_2 = (n, S, R \cup \{j\}, \perp)$ is also valid; hence, the statement $Q(n)$ also trivially holds.

Consider some $1 \leq r < n$ such that for all $r + 1 \leq t \leq n$, the statements $P(t)$ and $Q(t)$ are true.

We first prove the statement $P(r)$. Suppose player $i \neq i_0$ is removed. Since the case $i \notin S$ is trivial, it suffices to consider $i \in S$. Suppose $j = O_i(r)$ is the r -th favorite item of player i . If $j \notin R$, then player i is not going to compete for any item in round r , and so this is equivalent to removing player i in round $r + 1$, and we can use $P(r + 1)$; hence, we can assume $j \in R$.

Suppose $C = \{s \in S : O_s(r) = j\}$ are survivors that compete for item j in this round r . Observe that survivors in S not competing for j behave the same in round r in states $\lambda = (r, S, R, \perp)$ and $\lambda_1 = (r, S \setminus \{i\}, R, \perp)$. There are two sub-cases.

- $|C| = 1$. This means in the current round r , no other survivor in S views j as its r -th favorite item. In particular, this implies that item j is not within the r most favorite items of player i_0 . We can construct a coupling between the states at the beginning of round $r + 1$ resulting from λ and λ_1 . For every state $\hat{\lambda} = (r + 1, \hat{S}, \hat{R}, a)$ that results from λ , we map it to $\hat{\lambda}_1 = (r + 1, \hat{S}, \hat{R} \cup \{j\}, a)$ that results from λ_1 ; observe that both transitions occur with the same probability. Hence, the statement $Q(r + 1)$ implies that $P(r)$ is true in this case.
- $|C| \geq 2$. We construct a coupling between states resulting from λ and λ_1 . Suppose from λ , the next state at the beginning of round $r + 1$ is $\hat{\lambda} = (r + 1, \hat{S}, \hat{R}, a)$. There are two further cases.
 - (i) Case $i \in \hat{S}$. This means player i did not win in AugTourn for item j . In this case, we map $\hat{\lambda}$ to $\hat{\lambda}_1 = (r + 1, \hat{S} \setminus \{i\}, \hat{R}, a)$, and use $P(r + 1)$ in this case.
 - (ii) Case $i \notin \hat{S}$. This means player i has won item j starting from state λ . To create the coupling under this case, from state $\hat{\lambda}_1$, we pick a player $i' \in C \setminus \{i\}$ uniformly at random to be the winner of item j . The result is the state $\hat{\lambda}_1 = (r + 1, \hat{S} \setminus \{i'\}, \hat{R}, a)$, and we also use $P(r + 1)$ for this case.

We next prove the statement $Q(r)$, i.e., we add item $j \notin R$ at the beginning of round r . Let r^* be the minimum round at least r such that there exists $i \in S$ such that $O_i(r^*) = j$. If $r^* > r$, then we can use the statement $Q(r^*)$;

hence, we can assume that there is some $i \in S$ such that $O_i(r) = j$, and so $C = \{i \in S : O_i(r) = j\}$ is non-empty.

We create a coupling between states at the beginning of round $r + 1$ resulting from $\lambda = (r, S, R, \perp)$ and $\lambda_2 = (r, S, R \cup \{j\}, \perp)$, respectively.

Observe that if $\hat{\lambda} = (r + 1, \hat{S}, \hat{R}, a)$ results from λ , then $C \subseteq \hat{S}$ because players in C did not compete for any item in round r . However, from state λ_2 , exactly one of C will win item j and be removed from S ; hence, we randomly pick one player $i \in C$ to win item j . There are two cases.

(i) Case $i = i_0$. Observe that player i_0 prefers item j to any item in R . Hence, in this case $\hat{\lambda}_2 = (r + 1, \hat{S} \setminus \{i_0\}, \hat{R}, j)$ is better for i_0 than $\hat{\lambda}$.

(ii) Case $i \neq i_0$. In this case, we consider $\hat{\lambda}_2 = (r + 1, \hat{S} \setminus \{i\}, \hat{R}, a)$, and apply the statement $P(r + 1)$.

This concludes the induction proof. □

Lemma 7 (Maximin Security). *The PP protocol in Algorithm 1 is maximin secure against a fail-stop adversary that controls up to $n - 1$ players.*

Proof. We show that an honest player i_0 cannot be harmed by a fail-stop adversary Adv. Specifically, we argue that if $v \in \Delta(\mathcal{M})$ is the distribution vector received by i_0 under the honest execution and v^{Adv} is the corresponding one under the strategy of Adv, then $v^{\text{Adv}} \succeq v$ with respect to the preference of i_0 .

Observe that Algorithm 1 consists of multiple instances of AugTourn. There are two cases.

- Suppose i_0 participates in an instance of AugTourn for some item j in some round r . Observe that at this moment, item j is the most preferred item among the remaining items by i_0 . Lemma 2 states that the probability that i_0 wins item j cannot be decreased by the adversary Adv. If the strategy of Adv causes any corrupted player to abort in this instance of AugTourn, the resulting distribution received by i_0 still dominates the original distribution.
- Suppose i_0 does not participate in an instance of AugTourn. Lemma 2 states that there exists a coupling between an honest execution and an execution under the strategy of Adv such that the survivor set S^{Adv} produced under Adv is always a subset of S under honest execution. Lemma 6 states that removing other players cannot harm the honest player i_0 . Hence, the resulting distribution received by i_0 under Adv still dominates that produced under an honest execution.

Performing a hybrid argument on every instance of AugTourn in Algorithm 1 gives the required result. □

6 Achieving Uniform Dominance

Re-Visiting PS mechanism. We can see from the proof of Theorem 4 that the hurdle in realizing the PS mechanism is that it can capture the coin-flipping

problem, for which a fail-stop adversary controlling at least $\frac{n}{2}$ players can cause a non-zero bias on the outcome probability (from the ideal $\frac{1}{2}$). Even though we do not know how to modify PS to achieve maximin security, it has inspired us to design a variant of the protocol that can achieve uniform dominance against Byzantine adversaries.

6.1 OnlinePSVar Protocol

Protocol Design Intuition. The *online probabilistic serial with varying rates* (OnlinePSVar) protocol in Algorithm 2 is based on the original PS mechanism, but as soon as an item is totally consumed, it is rounded via the lottery problem according to the fractional consumptions by the players. Observe that to make the description intuitive, the consumption of an item seems to be an “action” by a player. However, this is actually performed automatically according to the input preference profile (about which a player could still lie though). The players actually only actively participate in instances of AugTourn, which is assumed to take zero “time” in the consumption process.

1	Input: A preference profile $\sigma \in \mathcal{S}^{\mathcal{N}}$ (declared by players) for the items \mathcal{M} .
2	Output: A (random) assignment $A \in \mathcal{A}$.
3	Initialization:
4	Set A to an empty assignment (i.e., a zero matrix).
5	All items in \mathcal{M} are unconsumed.
6	for each player $i \in \mathcal{N}$, set consumption rate of $s_i \leftarrow 1$ unit of item per unit time.
7	Consider each item as 1 unit of an infinitely divisible commodity; initialize time $= 0$.
8	while $\exists i \in \mathcal{N} : s_i > 0$ do
9	Every player $i \in \mathcal{N}$ consumes its favorite item (according to its preference order σ_i) that is still not totally consumed at rate s_i .
10	When an item $j \in \mathcal{M}$ is totally consumed by some subset $\widehat{\mathcal{N}}$ of players (if there is more than one such item, process each item independently), do the following:
11	Let $\mathbf{p} = (p_i : i \in \widehat{\mathcal{N}})$ describe how players fractionally divide item j .
12	Run AugTourn(\mathbf{p}) to obtain the winner \widehat{i} and the survivor set $S \subseteq \widehat{\mathcal{N}} \setminus \{\widehat{i}\}$; recall that a survivor is a non-winning player that is not identified as corrupted in AugTourn.
13	Assign $A(\widehat{i}, j) \leftarrow 1$.
14	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <i>Varying Rates:</i> for $i \in S$, set $s_i \leftarrow s_i + s_{\widehat{i}} \times \frac{p_i}{\sum_{k \in \widehat{\mathcal{N}} \setminus \{\widehat{i}\}} p_k}$. </div>
15	for $i \in \widehat{\mathcal{N}} \setminus S$, set rate $s_i \leftarrow 0$.
16	end
17	Any remaining items are arbitrarily assigned to players with no items yet (according to some pre-determined rule) and update A accordingly.
18	return assignment A

Algorithm 2: OnlinePSVar Protocol

We would like to highlight an important feature: when line 14 “Varying Rate” is executed, the rate of the winner in AugTurn is distributed among survivors proportional to their winning probabilities. We shall see that this is extremely important to achieve uniform dominance, as illustrated later in Example 3.

Terminology. Strictly speaking, OnlinePSVar is a protocol. However, when we say the OnlinePSVar mechanism, we mean the corresponding mapping that takes an input preference profile and returns a distribution of assignments when all players behave honestly in the protocol.

Lemma 8. (Obvious Properties). *The OnlinePSVar mechanism achieves strong equal treatment and stability.*

Proof. Recall that to consider the properties of the mechanism, we investigate what happens when all players are honest.

From the description in Algorithm 2, for two players with exactly the same preference for their top k items, before all those k items are totally consumed or one of them is assigned an item, they will behave in exactly the same way. This implies that the mechanism achieves strong equal treatment.

Stability is also obvious because the first item that is totally consumed will be the top choice for the player that receives it. Applying this observation repeatedly to the remaining items gives the conclusion. \square

Varying Rates. Observe that the re-distribution of consumption rates among survivors after AugTurn and the detection of corrupted players can make the process very complicated. Surprisingly, we have observed the following structural property of the process. In retrospect, we could have replaced line 14 “Varying Rates” with a much simpler updating rule, but this will make the description less intuitive.

Lemma 9 (Consumption Rate at Joining Time). *Consider the consumption process in OnlinePSVar. Suppose a player starts the consumption of an item at time $t \geq 0$ (with a positive rate). Then, we must have $t < 1$ and its consumption rate for that item is $\frac{1}{1-t}$. This holds even when all players are controlled by a Byzantine adversary.*

Proof. Observe that the consumption process can be affected by the outcome of each AugTurn instance (which has at most n outcomes). Hence, there are at most n^n scenarios of the process, in which each scenario has at most n possible times that a player can join the consumption of an item. Therefore, there are only a finite number of times a player can join the consumption of an item, and we can prove the result by induction on the joining time.

The base case $t = 0$ is trivial, because initially all players have a consumption rate of 1. For the induction hypothesis, suppose that some player i joins the consumption of an item j at some time $t > 0$ such that if any player starts joining the consumption of any item at time $t' < t$, it must be the case that $t' < 1$ and the rate at that moment is $\frac{1}{1-t'}$.

Since $t > 0$, this means that player i has just finished participating in some AugTurn for another item \hat{j} , and is a (non-winning) survivor. Consider the winner d for item \hat{j} , and let t_d and t_i be the corresponding joining times.

By the induction hypothesis, both t_d and t_i are less than 1, and the two players are consuming item \hat{j} at rates $s_d = \frac{1}{1-t_d}$ and $s_i = \frac{1}{1-t_i}$. This means that in the instance of AugTurn for item \hat{j} , their winning probabilities are $p_d = (t - t_d)s_d$ and $p_i = (t - t_i)s_i$.

Therefore, the rate re-distribution rule gives that after AugTurn, the new rate for player i is:

$$s = s_i + s_d \cdot \frac{p_i}{1-p_d} = s_i + \frac{s_d \cdot (t-t_i)s_i}{1-(t-t_d)s_d} = s_i \cdot \frac{1-(t-t_d)s_d + s_d \cdot (t-t_i)}{1-(t-t_d)s_d} = \frac{1}{1-t}.$$

Since $s > 0$, we must have $t < 1$ and the inductive step is completed. Observe that no assumption about truthfulness or honesty is needed in this proof. \square

Corollary 2 (Same New Rate for Survivors of AugTurn). *In OnlinePSVar, any instance of AugTurn with more than one candidate is completed strictly before time 1, and all the resulting survivors have the same new consumption rates. This holds even when all players are controlled by a Byzantine adversary.*

6.2 OnlinePSVar Achieves Uniform Dominance

We first illustrate that the variant OnlinePS (which is the variant of Algorithm 2 with line 14 “Varying Rates” removed) does not achieve uniform dominance (even when all players are honest).

Example 3 (OnlinePS does not achieve uniform dominance). Consider $n = 5$ with $A = \{1, 2, 3\}$ and $B = \{4, 5\}$, where m_1 and m_2 are the top two items for all players, but $m_1 \succ_A m_2$ and $m_2 \succ_B m_1$. Then, a simple calculation shows that the probability that player 1 receives item m_1 or m_2 is: $\frac{1}{3} + \frac{2}{3} \times \frac{1}{12} < \frac{2}{5}$.

Interrupted Process and Claiming Ownership in OnlinePSVar. Later in our proofs, we would like to consider the probability of whether an honest player i_0 has already been assigned an item by some time $t \in [0, 1]$. However, it is possible that at time t , a player is still consuming some item j . We introduce the concept of *claiming ownership*. The interpretation here is that we interrupt the process at this time, and sample a Bernoulli random variable with the parameter equal to the fraction of item j already consumed by player i to determine whether player i_0 should receive that item. Hence, when we say that a player has *claimed* the ownership of an item by time t , we mean that the player has either received that item or has a claim to that item via the above Bernoulli process. Observe that for $t = 1$, the claim of ownership and the actual assignment are equivalent.

Lemma 10 (Probability of Ownership). *Suppose an honest player i_0 is about to start consuming some item at time t_0 (which means at this point i_0 still has not received any item). Then, for all $t \in [t_0, 1]$, the probability that i_0 will have claimed ownership of any item by time t is $\frac{t-t_0}{1-t_0}$.*

Proof. As argued in Lemma 9, the set of possible times that some player starts to consume some item is finite. Hence, we prove the result by (backward) induction starting from larger values of t_0 .

Suppose t_0 is the largest time that any player can start to consume an item, and player i_0 happens to start consuming an item at time t_0 . By Lemma 9, player i_0 is consuming that item at rate $\frac{1}{1-t_0}$. The maximality of t_0 means that this is the last ever item available to i_0 and there cannot be another player competing with i_0 ; otherwise, it would have been possible to start consuming another item at a time later than t_0 . Hence, by time $t \in [t_0, 1]$, player i_0 would have consumed $\frac{t-t_0}{1-t_0}$ fraction of item j , which is exactly the probability of claiming ownership by the Bernoulli process at this moment.

For the induction hypothesis, suppose an honest player i_0 starts to consume some item j at some time t_1 such that for all $t_0 > t_1$, the required result holds if an honest player starts to consume an item at time t_0 .

By Lemma 9, i_0 consumes item j at rate $\frac{1}{1-t_1}$. Observe that if item j is not fully consumed by time t , then the same argument as the base case holds, and the probability that i_0 claims ownership of j by time t is $\frac{t-t_1}{1-t_1}$.

Otherwise, we let the consumption process carry on until item j is fully consumed at some time $t_0 > t_1$. By Lemma 9, at time t_0 , the fraction of item j consumed by i_0 is $\frac{t_0-t_1}{1-t_1}$, which is the probability that (honest) player i_0 will win in AugTourn for item j . However, conditioned on i_0 losing item j at time t_0 , the induction hypothesis says that by time $t > t_0$, i_0 would have claimed ownership of some item with probability $\frac{t-t_0}{1-t_0}$.

Hence, to summarize the case when item j is fully consumed before time t , the probability that i_0 will have claimed ownership of some item by t is:

$$\frac{t_0-t_1}{1-t_1} + (1 - \frac{t_0-t_1}{1-t_1}) \cdot \frac{t-t_0}{1-t_0} = \frac{t-t_1}{1-t_1}.$$

This completes the induction proof. □

Lemma 11 (Uniform Dominance). *The protocol OnlinePSVar achieves uniform dominance.*

Proof. It suffices to show that for all $\ell \in [n]$, the probability that an honest player i_0 will receive an item among its top ℓ favorite items is at least $\frac{\ell}{n}$, no matter whether the other players are truthful or honest.

Observe that the sum of consumption rates over all players is at most n (when a corrupted player is detected, its rate is set to 0). Therefore, before $t = \frac{\ell}{n}$, at least one of player i_0 's top ℓ items has not been totally consumed. This means that before time t , player i_0 is consuming only among its top ℓ items, and it is obvious that a player can only claim ownership of an item that it has attempted to consume.

Hence, by Lemma 10 with $t_0 = 0$, the probability that player i_0 has claimed ownership of one of its top ℓ items by time $t = \frac{\ell}{n}$ is $\frac{\ell}{n}$. Finally, consider the natural coupling between the interrupted process at time t and the original OnlinePSVar process (achieved by coupling the Bernoulli process at interruption with the corresponding AugTourn) such that any item whose ownership is claimed by i_0 at time t is also received by i_0 in OnlinePSVar. This completes the proof. □

6.3 OnlinePSVar Is Not Maximin Secure nor Strongly Truthful

We will give two examples to show that OnlinePSVar is neither maximin secure nor strongly truthful.

OnlinePSVar is not Maximin Secure. To show that OnlinePSVar cannot achieve maximin security even against fail-stop adversary, we first illustrate the idea of constructing the counter-example for showing OnlinePSVar is not maximin secure. Suppose there are two sets of players X_1, X_2 with large enough $|X_1|$ and $|X_2|$, and three special players, i_1, i_2, i_3 . The following is the profile regarding the three special players with unimportant items omitted:

$$\begin{aligned}
 & m_3 \succ_1 \cdots \succ_1 \cdots \succ_1 \cdots \\
 & m_1 \succ_2 m_2 \succ_2 m_3 \succ_2 \cdots \\
 & m_1 \succ_3 m_2 \succ_3 m_3 \succ_3 \cdots
 \end{aligned}$$

We assume players in X_1 like m_1 the most and will compete for m_1 at time 0, and later they will not interfere with items m_2 and m_3 . For players in X_2 , we assume that they will compete for m_2 at time $\frac{1}{2}$, but they will not interfere with m_1 and m_3 at all. We have two observations.

1. We consider the game with players $X_1 \cup \{i_1, i_2, i_3\}$. In this case, at time 0, i_2, i_3 and players in X_1 will compete for m_1 , and it will be finished in a short time. Since i_2 and i_3 have low chances of getting m_1 , they will then compete for m_2 and finish this round at time $\approx \frac{1}{2}$. As a result, one of the players will get m_2 and the other will join in competing for m_3 with i_1 at time $\approx \frac{1}{2}$.
2. We consider the game with players $X_2 \cup \{i_1, i_2, i_3\}$. In this case, one of i_2 and i_3 will get m_1 and the other (namely, i_2) will start to compete for m_2 at time $\frac{1}{2}$. Note that players in X_2 will join as well and item m_2 will be finished in a short time. Since i_2 has low chance for getting m_2 , it will join in competing m_3 with i_1 at time $\approx \frac{1}{2}$.

The above observations illustrate a fact that, when one of X_1 or X_2 is absent, the probability for i_1 getting m_3 in these two cases could be arbitrarily close. Hence, we consider a sequence of games constructed by replacing participants involved in the game: we start from the game described in observation 1, and gradually shift to the one in observation 2 by adding players from X_2 or removing players from X_1 . Then, there exist two consecutive games, where, the probability of i_1 getting m_3 declines after removing some player from X_1 , or increases after adding some player from X_2 . This gives the intuition for the following counter-example.

Example 4 (OnlinePSVar Not Maximin Secure). Consider the preference profile given by the following matrix, where each row corresponds to a player and each entry (i, j) contains the index of the j -th favorite item for player i .

3	1	2	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	14	13	12	11	10	9	8	7	6	5	4	2	3
4	2	14	13	12	11	10	9	8	7	6	5	1	3
4	2	14	13	12	11	10	9	8	7	6	5	1	3
5	2	14	13	12	11	10	9	8	7	6	4	1	3
5	2	14	13	12	11	10	9	8	7	6	4	1	3
6	2	14	13	12	11	10	9	8	7	5	4	1	3
6	2	14	13	12	11	10	9	8	7	5	4	1	3
7	2	14	13	12	11	10	9	8	6	5	4	1	3
7	2	14	13	12	11	10	9	8	6	5	4	1	3
8	2	14	13	12	11	10	9	7	6	5	4	1	3
8	2	14	13	12	11	10	9	7	6	5	4	1	3

The player 4 acts as X_1 and players $5, \dots, 14$ act as X_2 in the discussion above. Using a program, we can use the brute force approach to compute the probability that a player wins a certain item. When all players act honestly, player 1 will get its favorite item m_3 with probability $\frac{1132927}{1499784} \approx 0.7553$. However, if player 4 aborts at the beginning, then agent 1 gets m_3 with probability $\frac{77}{102} \approx 0.7549$, which violates maximin security.

OnlinePSVar is not Strongly Truthful. The following example 5 is a counterexample to show that OnlinePSVar is not strongly truthful.

Example 5 (OnlinePSVar Not Strongly Truthful). Consider $n = 4$ players with the following true preferences:

$$\begin{aligned}
 m_1 &\succ_1 m_2 \succ_1 m_3 \succ_1 m_4 \\
 m_1 &\succ_2 m_2 \succ_2 m_3 \succ_2 m_4 \\
 m_2 &\succ_3 m_3 \succ_3 m_4 \succ_3 m_1 \\
 m_2 &\succ_4 m_3 \succ_4 m_4 \succ_4 m_1
 \end{aligned}$$

In OnlinePSVar, player 2 obtains one of its top two items $\{m_1, m_2\}$ with probability $\frac{1}{2}$. We show that player 2 can increase this probability by lying about its preference as:

$$m_2 \succ m_1 \succ m_3 \succ m_4.$$

At time $t = \frac{1}{3}$, item m_2 is totally consumed.

If player 2 loses the tournament for item m_2 , then it will compete with player 1 for item m_1 starting at time $t = \frac{1}{3}$. In OnlinePS, the rate of player 2 remains 1, while in OnlinePSVar, its rate is increased to 1.5. Hence, it suffices to do the calculation for the former case, in which item m_1 will be totally consumed at time $t' = \frac{2}{3}$, when player 2 will get $\frac{1}{3}$ fraction of m_1 .

Hence, by lying, the probability that player 2 obtains either m_1 or m_2 is: $\frac{1}{3} + (1 - \frac{1}{3}) \cdot \frac{1}{3} = \frac{5}{9}$, which is larger than before.

7 Conclusion

We have considered the game-theoretic notion of maximin security for protocols solving the ordinal assignment problem, where randomness is necessary to achieve the fairness notion of equal treatment.

Our major contribution is the impossibility result that shows no maximin secure protocol can satisfy both strong equal treatment and ordinal efficiency, thereby also excluding the possibility of any maximin secure protocol that realizes the well-known PS mechanism. However, the problem of whether there exists a maximin secure protocol that realizes RP is still open. In general, the following questions are interesting future directions.

- Does there exist a maximin secure protocol that achieves both strong equal treatment and uniform dominance?
- Does there exist a maximin secure protocol that achieves both strong equal treatment and truthfulness?

References

1. Abdulkadiroglu, A., Sönmez, T.: Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica* **66**(3), 689–702 (1998). <https://EconPapers.repec.org/RePEc:ecm:emetrp:v:66:y:1998:i:3:p:689-702>
2. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: a composable treatment. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 324–356. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_11
3. Blum, M.: How to exchange (secret) keys. *ACM Trans. Comput. Syst.* **1**(2), 175–193 (1983)
4. Bogomolnaia, A., Moulin, H.: A new solution to the random assignment problem. *J. Econ. Theory* **100**(2), 295–328 (2001)
5. Chung, K.-M., Chan, T.-H.H., Wen, T., Shi, E.: Game-theoretic fairness meets multi-party protocols: the case of leader election. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 3–32. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_1
6. Chung, K.-M., Guo, Y., Lin, W.-K., Pass, R., Shi, E.: Game theoretic notions of fairness in multi-party coin toss. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018. LNCS, vol. 11239, pp. 563–596. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03807-6_21
7. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: STOC, pp. 364–369. ACM (1986)
8. Echenique, F., Immorlica, N., Vazirani, V.V.: *Online and Matching-Based Market Design*. Cambridge University Press, London (2021)
9. Gärdenfors, P.: Assignment problem based on ordinal preferences. *Manag. Sci.* **20**(3), 331–340 (1973). <https://EconPapers.repec.org/RePEc:inm:ormnsc:v:20:y:1973:i:3:p:331-340>
10. Hylland, A., Zeckhauser, R.: The efficient allocation of individuals to positions. *J. Political Econ.* **87**(2), 293–314 (1979). <https://EconPapers.repec.org/RePEc:ucp:jpolec:v:87:y:1979:i:2:p:293-314>

11. Lin, H., Pass, R.: Constant-round nonmalleable commitments from any one-way function. *J. ACM* **62**(1), 1–30 (2015)
12. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: FOCS, pp. 120–130. IEEE Computer Society (1999)
13. Miller, A., Bentov, I.: Zero-collateral lotteries in bitcoin and ethereum. In: EuroS&P Workshops, pp. 4–13. IEEE (2017)
14. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2009). <http://www.bitcoin.org/bitcoin.pdf>
15. Pass, R.: Bounded-concurrent secure multi-party computation with a dishonest majority. In: STOC, pp. 232–241. ACM (2004)
16. Saban, D., Sethuraman, J.: The complexity of computing the random priority allocation matrix. *Math. Oper. Res.* **40**(4), 1005–1014 (2015). <https://EconPapers.repec.org/RePEc:inm:ormoor:v:40:y:2015:i:4:p:1005-1014>
17. Zhou, L.: On a conjecture by gale about one-sided matching problems. *J. Econ. Theory* **52**, 123–135 (1990). [https://doi.org/10.1016/0022-0531\(90\)90070-Z](https://doi.org/10.1016/0022-0531(90)90070-Z)



A New Approach to Garbled Circuits

Anasuya Acharya^{1(✉)}, Tomer Ashur^{2,3}, Efrat Cohen¹, Carmit Hazay¹,
and Avishay Yanai⁴

¹ Bar-Ilan University, Ramat Gan, Israel
{acharya,efrat.choen,carmit.hazay}@biu.ac.il

² imec-COSIC, KU Leuven, Leuven, Belgium
tomer.ashur@esat.kuleuven.be

³ Cryptomeria, Leuven, Belgium
tomercryptomeria.tech

⁴ VMware Research, Herzliya, Israel
yanai@vmware.com

Abstract. A garbling scheme is a fundamental cryptographic building block with a long list of applications. The study of different techniques for garbling a function, towards optimizing computation and communication complexity, has been an area of active research. Most common garbling techniques work by representing each gate in the circuit as a set of ciphertexts that encrypt its truth table row-by-row.

In this work we present a new garbling scheme in the random oracle (RO) model that garbles circuits in the gate-by-gate paradigm by capturing the gate functionality (AND, XOR) *as a whole* rather than as a set of ciphertexts. The final gate garbling requires 4κ bits of communication in expectation, 4 RO calls for garbling and 1 RO call for evaluation. We prove that the scheme satisfies privacy in the non-programmable random oracle model and against PPT adversaries. We also show how this scheme can be extended to support free-XOR and garble *any* gate functionality over binary inputs.

Keywords: Garbled Circuits · Gate-by-Gate Garbling · Random Oracles

1 Introduction

The theory and practice of garbling circuits has been the focus of a long line of research, starting from the seminal work of [Yao86], and further optimized in the works of [LP09, PSSW09, BHR12, ZRE15, HK20, RR21], to name a few. Garbled circuits (GCs) are a fundamental building block that represents a function and a secret input in such a way that evaluating the garbled circuit on the input representation reveals nothing beyond the function output. GCs have a long list of applications like constant round secure two-party computation (2PC) [LP09], constant round multiparty computation [BMR90, BLO16], zero-knowledge proofs [FNO15, GKPS18], bootstrapping obfuscators [App13], functional encryption [GKP+13], and verifiable computation [GGP10].

Owing to its wide range of applications, Bellare *et al.* presented an abstraction for garbling in [BHR12], viewing it as a fundamental building-block for use in cryptographic protocols. This abstraction is termed as a garbling scheme and is a framework defining four algorithms. A garbling algorithm takes a function representation, i.e. a circuit, and uses it to create a garbled circuit (GC). Depending on the scheme, the GC may have certain function hiding properties: given a GC, the actual functionality garbled remains hidden. The garbling algorithm also creates an input encoding function. Next, an input encoding algorithm takes any valid input to the circuit garbled and uses the input encoding function to give ‘input labels’ that correspond to the GC. The input labels typically have the property that, when looked at in isolation, they do not reveal the input represented. An evaluation algorithm takes a GC and a set of input labels for a certain input, and derives a representation of the function output. Finally, an output decoding algorithm derives the function output from its representation output by the evaluation algorithm. It is required that nothing beyond the function output is revealed. [BHR12] also gives various definitions of desirable properties for garbling schemes like correctness, privacy, authenticity and obliviousness.

A scheme for garbling circuits was first proposed in [Yao86] and its security was formalized in [LP09]. The formalism of [BHR12] captures this construction and many subsequent works in garbling published after [BHR12] have followed the same line of thought as [LP09], also describing themselves in terms of [BHR12]. [LP09] garbles a circuit in a gate-by-gate manner where each gate is garbled by encoding its truth table row-by-row, creating a set of ciphertexts. Subsequent optimizations reduce the size of garbled gates by either reducing ciphertext sizes, allowing certain ciphertexts to not require communication [PSSW09], or re-writing the gate functionality so that its truth table has fewer rows [ZRE15].

1.1 Our Contributions

We propose a novel scheme for garbling circuits in the gate-by-gate paradigm that captures the gate’s truth table as a whole in one encoding, rather than as a set of encrypted rows. We operate in the non-programmable random oracle (RO) model wherein both the garbler and the evaluator are given access to a common random oracle. Our garbling approach requires 4 RO queries to garble *any* binary gate functionality and 1 RO query for evaluation. For a computational security parameter κ , letting the length of each input label be κ , the expected length of each garbled gate is 4κ bits. We also describe how this scheme can be modified to support free-XOR at the cost of increasing the size of other garbled gates.

Although this scheme does not improve upon the current state-of-the-art in garbling size, it produces a garbling with size that is comparable. It also has certain advantages over schemes that produce garblings. For instance, the garbling scheme in [ZRE15] produces gate garblings of size 2κ while providing free-XOR compatibility. However, evaluating their GC requires 2 calls to the underlying cryptographic primitive for a gate, while our scheme requires only 1 RO call. The garbling scheme in [PSSW09] also produces gate garblings of size 2κ while making 4 calls to the underlying cryptographic primitive for garbling

and 1 call for evaluation. However, it does not support free-XOR and nor can it be extended to support it like our scheme allows. We also have an advantage in computation complexity over [RR21] that produces a gate garbling of size 1.5κ at the cost of up to 6 primitive calls for garbling and 3 for evaluation. Further novelty of our scheme lies in the new approach employed for garbling that opens up a variety of avenues for future work.

Our scheme satisfies *correctness* and *privacy* [BHR12] against a PPT adversary with access to $t(\kappa)$ queries to the random oracle for any polynomial $t(\cdot)$. Informally, the privacy-by-indistinguishability property requires that for two circuits C_0 and C_1 that have the same topology, and for two inputs x_0 and x_1 such that $C_0(x_0) = C_1(x_1)$, a garbling of C_0 along with input labels corresponding to x_0 should be indistinguishable from a garbling of C_1 and input labels for x_1 .

[BHR12] also contains a result stating that if the *leakage function* for a garbling scheme is invertible, the definitions of privacy-by-indistinguishability and privacy-by-simulation are equivalent. For our garbling scheme, the leakage function – information about the function revealed by the garbling – is the topology of the circuit garbled. This is indeed an invertible leakage – given a circuit topology, one can construct a circuit that has that topology. Therefore, it holds that our garbling scheme also satisfies privacy-by-simulation.

1.2 Related Work

Secure garbling of circuits and corresponding ways of succinctly representing the garbling has been the aim of a long line of research [BMR90, NPS99, KS08, LP09]. The most common paradigm for garbling a circuit operates at the gate level (also known as the ‘gate-by-gate paradigm’) where for each gate in the circuit, each line in the truth table of the gate functionality is encrypted separately. The underlying primitive for encryption is a symmetric-key algorithm (*e.g.*, a pseudorandom function (PRF), a circular-correlation robust hash function (CCR), a CPA-secure dual-key cipher (DKC)) which yields extremely fast algorithms. This paradigm led to a long sequence of successful optimizations in computation and communication, that established garbled circuits as a practical tool for achieving 2PC [PSSW09, KMR14, ZRE15].

Minimizing the size of garbled circuit representation so as to reduce the communication complexity is a widely studied research area. To this effect, [KS08] proposes a garbling technique that allows for ‘free-XOR’ – an XOR gate need not be represented in the garbling at all. Following [LP09, PSSW09] proposes schemes that garble each gate in a circuit by garbling its truth-table row wise, but in a way that certain garbled rows need not be communicated. For a computational security parameter κ , one such scheme (GRR3) produces a gate garbling of size of 3κ , while still remaining compatible with free-XOR. Another scheme (GRR2) garbles each gate with 2κ -bits, at the cost of forfeiting free-XOR compatibility. Both of these are improvements over the 4κ -bits required in [LP09]. Another work, [ZRE15] takes this further by proposing a garbling technique that garbles each gate using 2κ -bits, while remaining compatible with free-XOR. [KKS16]

shows a scheme in which 2κ bits can be used to garble internal gates of a circuit, while gates with circuit input wires as input can be garbled using κ bits. For certain classes of circuits, formulas in particular, their construction requires between κ and 1.5κ bits per garbled gates on average.

Table 1. Comparison of our scheme with related work. We compare the garbling size and the number of primitive calls for garbling and evaluation for an AND gate. κ is the computational security parameter.

Garbling Scheme	Gate Garbling Size	Number of RO Calls	
		Garbling	Evaluation
[ZRE15](Free-XOR)	2κ	4	2
[RR21](Free-XOR)	$1.5\kappa + 8$	6	3
Our Work	4κ	4	1
Our Work (Free-XOR)	8κ	4	1

The state-of-the-art in garbled gate size optimization today is [RR21] where the size of each garbled gate is 1.5κ bits. Pursuing a different line of garbling size optimization, [HK20] proposes a scheme that reduces the size of the circuit as a whole to the size of the longest *branch* of computation.

An extended line of works the generalizes garbling is the study of *randomized encodings* [IK00, Ish13, App17]. Given a function f and an input x , a randomized encoding is a representation $\hat{f}(x, r)$ generated using randomness r such that no information beyond $f(x)$ can be derived from it. A garbling can be viewed as a special case of a randomized encoding. Specifically, a projective garbling such as ours is a case of a *decomposable* randomized encoding, where given the garbling and the active input labels only, nothing beyond the function output is revealed.

2 Technical Overview

Our garbling scheme operates in the random oracle model where both the garbler and evaluator get access to a random oracle (RO). Below we discuss the key design aspects of the core scheme. Discussion about the free-XOR extension is deferred to Sect. 5.

The Garbling Algorithm. Conforming to the [BHR12] formalism, the input to the garbling algorithm is a circuit \mathbf{C} ; and it outputs a garbled circuit F , an input encoding set e , and an output decoding set d . The algorithm itself can be separated into the following subroutines that are executed sequentially: (1) $\text{Init}(\mathbf{C}) \rightarrow e$; (2) $\text{Circuit}(\mathbf{C}, e) = (F, D)$; (3) $\text{DecodingInfo}(D) \rightarrow d$.

Input Label Sampling. The first subroutine in the garbling algorithm takes the circuit \mathbf{C} and creates the input encoding set e . This subroutine $\text{Init}(\cdot)$ is a randomized algorithm. From within \mathbf{C} , this algorithm only uses n , the number of input wires. This allows the generation of e potentially ahead of knowing the function f . Similar to other traditional garbling schemes, the scheme we design is also a projective garbling scheme. So e contains a set of input wire labels. In our construction, for each of the n input wires, for an ‘external length parameter’ ℓ , an ℓ -length label is sampled uniformly at random to represent the 0 and 1 bit, under the constraint that both labels for the same wire cannot be the same.

Gate-by-Gate Garbling. The next subroutine $\text{Circuit}(\cdot)$ is a deterministic function. It takes the input encoding set e with all the randomness it entails, and extends it to create the complete garbled circuit F and output wire labels D . In order to extend the existing randomness in a way that lets the garbling preserve its privacy, $\text{Circuit}(\cdot)$ makes black box calls to a random oracle RO.

Each gate in the circuit is garbled separately and in a topological order. To this effect, for the q total gates in the circuit \mathbf{C} , each gate is assigned an index g in this ordering. The random oracle RO employed throughout the gate-by-gate garbling process is tweakable: it takes as an additional input the gate index g so that it behaves independently for each gate.

Garbling a Gate. For a gate g , let A and B be its input wires, g be its output wire index, and f_g be its functionality (e.g., AND, XOR). When garbling a gate, our methods deviate significantly from traditional garbling techniques. At its core, we make the following observation: each gate is a binary gate so there are 4 combinations of input values, but only two possible output values corresponding to one output wire.

Therefore, at its core, a gate garbling is a means to *convert* a pair of input labels into an output label. For a wire A , L_0^A and L_1^A are its labels (similarly L_1^B, L_0^B for B , and L_1^g, L_0^g for output wire g). We require that for the gate g , the input label combinations be mapped to (L_0^g, L_1^g) in such a way that the gate functionality f_g is preserved. For instance, if the gate is an AND gate, $\{(L_0^A, L_0^B), (L_0^A, L_1^B), (L_1^A, L_0^B)\}$ should be mapped to L_0^g , and (L_1^A, L_1^B) to L_1^g .

We encode all four input label pairs into *one encoding* ∇^g such that, given one label from each input wire, ∇^g can be used to *convert* these into the correct output label. The details on how ∇^g is generated can be found in Appendix A where Table 3 indicates how the garbling for the AND functionality is generated and Table 4 indicates the same for the XOR functionality. These tables are part of the description of the garbling scheme: that is, they are predetermined and remain the same regardless of the circuit garbled or the randomness used.

The entire gate garbling process is a result of deterministic steps starting from the input label values. For gate g with input labels L_1^A, L_0^A and L_1^B, L_0^B , first, in order to eliminate redundancy, for each pair of input bits $(a, b) \in \{0, 1\}^2$ the input labels is input to a random oracle: $\text{RO}^g(L_a^A, L_b^B) \rightarrow X_{ab}^g$. The random oracle RO takes as input the tweak g and two labels with total length 2ℓ , and outputs an ℓ' -length string. The output length ℓ' is much larger than that of the input

and each RO output string is sampled uniformly at random and independently of the responses of other queries to RO^g .

Next, the random oracle outputs $(X_{00}^g, X_{01}^g, X_{10}^g, X_{11}^g)$ are used to derive a single ℓ^g -bit string ∇^g (that is padded by 0s to make its length equal to ℓ') that encodes the gate functionality. ∇^g has the properties that given any one X_{ab}^g , it maps it to an ℓ -bit uniformly random binary string $L_{f_g(a,b)}^g$. The gate garbling ∇^g has Hamming weight ℓ and the positions in this string that contain ‘1’ are termed as ‘effective key positions’. Each bit in ∇^g is set independently until its hamming weight becomes ℓ . We denote the length of the garbling up to this point as ℓ^g bits. It follows that ℓ^g varies for each gate g , but it still holds that $\ell^g = O(\kappa)$. The mapping of X_{ab}^g to an output label is done by projecting the bits in X_{ab}^g over the effective key positions in ∇^g . The resulting output label is of length ℓ and is independently and identically distributed (i.i.d.) over all bit positions. It also holds that the pair ∇^g and X_{ab}^g do not reveal any information about the inactive output label or the other random oracle outputs.

Decoding Information. Once all the garbled gates and output wire labels are derived in F , it remains to generate the output decoding information d . In our construction, we employ another random oracle RO' for this. In the subroutine that creates the decoding information, for every output wire j , we sample an ℓ -bit string d^j . This string has the property that, given output wire labels (L_0^j, L_1^j) , it holds that $\text{RO}'(L_0^j, d^j) = 0$ and $\text{RO}'(L_1^j, d^j) = 1$. Note that such a decoding will always yield some output even for arbitrary ℓ -bit strings that are not output labels. The subroutine $\text{DecodingInfo}(D) \rightarrow d$ generates this decoding information given the output wire labels set.

Evaluating the Garbled Circuit. An evaluator, given the garbled circuit F , a set of input wire labels X , and the decoding information d , works gate-by-gate. It has access to RO and RO' and knows the indices of each gate. Starting with the input labels $L \in X$ we term each value in its view during an honest evaluation as *active*. For each gate g , with active input labels L_a^A, L_b^B , the evaluator works by first deriving $\text{RO}^g(L_a^A, L_b^B) = X_{ab}^g$. Then using X_{ab}^g and $\nabla^g \in F$, it computes $L_{f_g(a,b)}^g = X_{ab}^g \circ \nabla^g$ where \circ is the operation selecting the bits in X_{ab}^g over the effective key positions in ∇^g . For an output wire label L_b^j , using $d^j \in d$, it computes $\text{RO}'(L_b^j, d^j) = b$ as the function output.

Security Intuition. Our scheme satisfies *privacy* against a PPT adversary. This notion is modeled as a game between the adversary and a challenger where the adversary first picks two circuits \mathbf{C}_0 and \mathbf{C}_1 of its choice such that they have the same topology. That is, letting Φ denote the leakage function revealing the topology of a circuit, it needs to hold that $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$. The adversary also chooses two inputs x^0 and x^1 such that $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$. The challenger picks a bit $b \in \{0, 1\}$, garbles \mathbf{C}_b and encodes the input x^b . It sends the resulting (F, X, d) to the adversary and then the adversary, making up to a polynomial number of queries to the random oracles, needs to output which bit b the challenger chose.

In order to understand why our scheme satisfies this notion of privacy, first note that the garbling (F, d) in the challenge, in isolation does not reveal any information about the circuits \mathbf{C}_0 or \mathbf{C}_1 , beyond Φ , the topology that is identical. This is because the garbling hides the gate functionality as each gate garbling originates from the same distribution regardless of the functionality.

Given the complete challenge (F, X, d) , an honest evaluation already reveals the complete ‘active path’ in the garbling. Our proof follows by proving that the knowledge of the active path gives the adversary no advantage at all in distinguishing. That is, given the complete challenge (F, X, d) and all honest queries, they are distributed independently of the bit b . Hence, we identify that learning elements in the ‘inactive paths’ is a prerequisite to privacy violation.

We formalize the notion of “learning a label” as making a random oracle query leading eventually to an inactive label. We identify what kind of queries lead to this and term them as bad events¹. There are three bad events that are triggered by a query to RO. In ‘Bad Event 1’, the adversary “guesses” a candidate input label to the gate input wire A and queries it to RO together with the active input label of wire B . ‘Bad Event 2’ is defined symmetrically for the input wire B of a gate and ‘Bad Event 3’ is defined when inactive candidates are queried on both input wires. They are all analysed similarly.

‘Bad Event 1’ is triggered by two sub-events. First, when the candidate is the *inactive input label* and the query outputs either the *active output label* or the *inactive output label*. Second, when the output from the query is a *valid output label* - the active or inactive output wire label used in the garbling - but the input label tested is *not the inactive input label*. This case is possible since the RO maps each input to an output value independently and uniformly at random. So a value that is not the inactive input label is mapped to a gate output label.

Intuitively, the resistance against this event comes due to the size of the set of candidate labels to be tested. Stemming from the fact that ℓ is appropriately set, and there is a unique active label L^g , it follows that there are $2^\ell - 1$ candidate labels for which the output of RO is unknown to the adversary. Beyond this information, any two labels within the set of possible labels are uniform and independent. This holds by construction because the labels of a wire are either sampled uniformly at random (input wire labels) or derived as projections of random oracle outputs (internal wire labels). The latter also results in a random ℓ -bit string owing to the fact that the random oracle outputs are sampled freshly and uniformly at random for each distinct domain value, and the nature of the gate garbling ∇^g used to select a subset of these bits. Within the set of candidate labels for a wire, there is always the inactive label - used in the garbling - triggering the bad event. However, possibly other ‘false positive’ label values may also trigger the bad event owing to the nature of the random oracle outputs. Considering multiple RO queries amounts to sampling without replacement. In essence, the security argument boils down to the fact that the set of candidate

¹ The proof technique we use is similar to [BHKR13] except that our adversary is PPT rather than query bounded. Since we do not assume anything about the adversarial strategy, this implies only that the bound on the number of queries is polynomial.

labels is sufficiently large so that the adversary cannot cover a non-negligible portion of it within their query budget. The advantage gained by making RO queries increases linearly in the number of queries and decreases exponentially in ℓ (Theorem 4). Thus, the adversary’s advantage is always negligible.

It remains to argue that when adversarial queries are made across different random oracles in different gates the bound on the probability of bad events remains unchanged. As a special case, let us consider two gates such that the output wire of one feeds into the other gate as input. First, note that the co-domain (all possible RO outputs) of the random oracle is $\{0, 1\}^{\ell'}$. However the domain is much smaller: $\{0, 1\}^{2^\ell}$. Due to this, the size of its range (the subset of the co-domain that is the set of actual RO outputs corresponding to all the RO inputs) is also upper-bounded by 2^{2^ℓ} . However, due to the properties of the random oracle, it is not possible to distinguish between its co-domain and range without querying the domain set. If a label in the range of RO is the inactive input label to the next gate, it will have triggered a ‘Bad Event’ as the inactive output label of the previous gate, ending the game. If the label is not the inactive input label, the adversary learns that further queries using this label as input are unnecessary, but gains no insight how to choose the candidate for the next query among the other $2^\ell - 1$ candidate input labels. Learning not to query this label to RO has cost the adversary a query in the previous gate hence it remains the case that “one query \rightarrow one discarded value”.

Therefore no advantage beyond what was learned directly by the query propagates between gates. This analysis holds without loss of generality when considering any number of gates in the circuit. We use this to bound the probability of encountering any of the three bad events, given $t(\kappa)$ queries to the random oracles and conclude that this is negligible in κ in our main result (Theorem 1).

3 Preliminaries

Table 2 contains a list of all the parameters with respect to which our garbling scheme is constructed. We denote by $\{0, 1\}$ the set containing 0 and 1, and by $\{0, 1\}^n$ the set of vectors of length n with each position containing 0 or 1. We use $[]$ to denote an empty array. For a vector V and $i \in \mathbb{N}$, we denote by $V[i]$ the element in the i^{th} position in the vector. When $[i]$ is used in isolation, it signifies the set of elements $1, \dots, i$.

Circuit Notation. For a function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$, let \mathbf{C} be its circuit representation. Let q be the number of gates in \mathbf{C} . Each gate $g \in [q]$ is defined by a gate functionality $f_g \in \{\text{AND}, \text{XOR}\}$, two input wires A, B and an output wire g where, $A, B, g \in [n + q]$ and topological ordering holds: $A, B < g$.

Random Oracles. The security proof of our scheme holds in the random oracle model, which abstracts a truly random function. Following the notation from [KL14], the random-oracle model posits the existence of a public, random function \mathcal{R} that can be evaluated only by “querying” an oracle – which can be thought of as a “black box” – returning $\mathcal{R}(x)$ when given input x .

Table 2. Table of Parameters

Parameter	Information
n	number of circuit input wires
m	number of circuit output wires
q	number of gates in the circuit
ℓ	(external length parameter) length of a wire label
ℓ'	(internal length parameter) length of approximate keys
ℓ^g	length of garbled gate ∇^g
κ	computational security parameter
s	number of adversarial random oracle queries

Definition 1 (Random Oracle). A random oracle RO is an interface for an oracle function $\mathcal{R} : \{0, 1\}^a \rightarrow \{0, 1\}^b$ that is sampled uniformly from the family of functions that map the domain of binary strings $\{0, 1\}^a$ into $\{0, 1\}^b$.

Garbling Scheme. [BHR12] abstracts garbling as a primitive containing four algorithms as given in Definition 2. In the definition, a function f is represented as a circuit \mathbf{C} . We also denote by $\Phi(\mathbf{C}) = (n, m, q, \{A, B, g\}_{g \in [q]})$ the topology of the circuit \mathbf{C} . Finally, $x \in \{0, 1\}^n$ denotes the function input and $y \in \{0, 1\}^m$ denotes the function output.

Definition 2 (Garbling Scheme [BHR12]). Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ be a function with circuit representation \mathbf{C} and κ be a computational security parameter. A garbling scheme $\text{GS} = (\text{Gb}, \text{En}, \text{De}, \text{Ev})$ has four PPT algorithms:

- $\text{Gb}(1^\kappa, \mathbf{C}) \rightarrow (F, e, d)$: returns a garbling F , input encoding set e , and output decoding set d .
- $\text{En}(e, x) := X$: returns the encoding X for function input x .
- $\text{Ev}(F, X) := Y$: returns the output labels Y by evaluating F on X .
- $\text{De}(Y, d) := \{\perp, y\}$: returns either the failure symbol \perp or a value $y = f(x)$.

These algorithms must satisfy the following properties:

- **Correctness:** For every κ , circuit \mathbf{C} and input x ,

$$\Pr[y = \mathbf{C}(x) : (F, e, d) \leftarrow \text{Gb}(1^\kappa, \mathbf{C}), X = \text{En}(e, x), Y = \text{Ev}(F, X), y = \text{De}(d, Y)] = 1$$

- **Privacy:** Let Algorithm 1 denote the actions of the challenger in an indistinguishability game. Let Φ be a leakage function representing the topology of a circuit. For every PPT adversary \mathcal{A} (with access to RO), for all circuits $\mathbf{C}_0, \mathbf{C}_1$ s.t. $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$ and every x^0, x^1 s.t. $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$ of the choice of \mathcal{A} , there exists a negligible function μ such that \mathcal{A} 's advantage is,

$$\text{Adv}(\kappa) = \left| \Pr[\mathcal{A}^{\text{RO}}(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1, F, X, d) = b] - \frac{1}{2} \right| < \mu(\kappa)$$

Algorithm 1. Privacy

```

1: proc Challenger( $\mathbf{C}_0, \mathbf{C}_1, x^0, x^1$ )
2:   if  $x^0, x^1 \notin \{0, 1\}^n$  or  $\Phi(\mathbf{C}_0) \neq \Phi(\mathbf{C}_1)$  or  $\mathbf{C}_0(x^0) \neq \mathbf{C}_1(x^1)$  return  $\perp$ 
3:    $b \leftarrow \{0, 1\}$ 
4:    $(F, e, d) \leftarrow \text{Gb}(1^\kappa, \mathbf{C}_b)$ 
5:    $X = \text{En}(e, x^b)$ 
6:   Return  $(F, X, d)$ 

```

4 The Scheme

In this section we present our garbling scheme. The scheme itself is presented in Sect. 4.1. We present in Sect. 4.2 the intuition behind why the scheme is correct. In Sect. 4.3, we discuss the security guarantee and outline the proof of security. A full proof is presented in Appendix B.

4.1 Garbling Algorithm

For a function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$, let \mathbf{C} be its circuit representation. The garbling algorithm has the following form:

Algorithm 2. Algorithm $\text{Gb}(1^\kappa, \mathbf{C})$

```

1: set the external length parameter  $\ell = \kappa$  and internal length parameter  $\ell' = 8\ell$ 
2:  $\text{Init}(\mathbf{C}, \ell) \rightarrow e$ 
3:  $\text{Circuit}(e, \mathbf{C}, \ell, \ell') = (F, D)$ 
4:  $\text{DecodingInfo}(D, \ell) \rightarrow d$ 
5: Return  $F, e, d$ 

```

The garbling algorithm as above begins by setting the variables ℓ and ℓ' defined in Table 2. These parameterize the lengths of the inputs and outputs of the random oracles employed in the construction. The ‘external length parameter’ ℓ parameterizes the length of all wire labels throughout the circuit. The additional ‘internal length parameter’ ℓ' parameterizes the length of the intermediate values in the gate garbling – the outputs of RO – and serves as a loose upper bound on the length of each gate garbling ℓ^g . The actual length of the gate garblings are variable and much smaller than ℓ' bits. Since the intermediate garbling values never have to be communicated, and so do not contribute to the communication complexity, ℓ' can be arbitrarily larger than ℓ . We refer the reader to Appendix A.1 for details as to why ℓ' is set to 8ℓ . Finally, ℓ is also the Hamming weight of ∇^g and parameterizes the effective length of the gate garbling. The complete garbling algorithm employs two random oracles of the following forms: (1) $\text{RO}^g : \{0, 1\}^{2\ell} \rightarrow \{0, 1\}^{\ell'}$; and (2) $\text{RO}' : \{0, 1\}^{2\ell} \rightarrow \{0, 1\}$. The first is used in each gate and so it uses the gate number g as a tweak. The latter is used for circuit output decoding.

Input Encoding Generation. The garbler starts by executing $\text{Init}(\mathbf{C}, \ell) \rightarrow e$, formally described in Algorithm 3. Let n be the number of input wires in \mathbf{C} and ℓ be the external length parameter. This algorithm uses the garbler’s randomness to sample ℓ -length labels to represent the 0 and 1 values for each input wire. These labels are sampled uniformly at random, under the constraint that two labels for the same wire cannot take the same value. This resulting set of input wire labels is the input encoding set e .

Algorithm 3. $\text{Init}(\mathbf{C}, \ell)$

- 1: extract n from \mathbf{C} and initialize $e = \square$
 - 2: **for** input wire $W \in [n]$ **do**
 - 3: Sample $L_0^W \leftarrow \{0, 1\}^\ell$ uniformly at random
 - 4: Sample $L_1^W \leftarrow \{0, 1\}^\ell - \{L_0^W\}$ uniformly at random
 - 5: Set $e[W] = e_W = (L_0^W, L_1^W)$
 - 6: **end for**
 - 7: **Return** e
-

Garbled Circuit Generation. The garbler now runs a deterministic algorithm to generate the garbled circuit: $\text{Circuit}(e, \mathbf{C}, \ell, \ell') = (F, D)$. This algorithm receives as input a circuit \mathbf{C} with q gates and a projective input encoding set e with labels for all n input wires. The output of this algorithm is a garbled circuit F , and a set D of pairs of labels for the m output wires of the garbled circuit. This is described in Algorithm 4. This algorithm works gate-by-gate where it creates a garbled gate by calling a subroutine described in Algorithm 5. The garbled circuit so produced is $F = (\nabla_1, \dots, \nabla_q)$.

Gate Garbling. We discuss now the subroutine that the garbling algorithm uses to garble each gate of the circuit: $(L_0^g, L_1^g, \nabla^g) \leftarrow \text{Gate}(L_0^A, L_1^A, L_0^B, L_1^B, g, f_g, \ell)$. This subroutine receives the gate index g , input labels set $(L_0^A, L_1^A, L_0^B, L_1^B)$ and a gate functionality indicator, $f_g \in \{\text{AND}, \text{XOR}\}$. For simplicity and completeness, we only discuss these functionalities although we can encode any gate functionality over binary inputs. The subroutine outputs a gate garbling ∇^g (with Hamming weight ℓ) and a set of labels for the gate output wire (L_0^g, L_1^g) , each of ℓ -bit length. The details of this subroutine are formally described in Algorithm 5. This is a deterministic function but with access to random oracle RO^g .

A gate is garbled in the following stages. First, given the set of input labels $(L_0^A, L_1^A, L_0^B, L_1^B)$, note that each of the combinations in $((L_0^A, L_0^B), (L_0^A, L_1^B), (L_1^A, L_0^B), (L_1^A, L_1^B))$ is a 2ℓ -bit string where ℓ bits are common with any other combination. To unlink the pairs, the input label combinations are passed into a random oracle RO^g . In order for this function to sample fresh outputs for different gates which may have potentially the same input wires, the input to RO^g also includes the gate id g as a *tweak*. For bits $a, b \in \{0, 1\}$, this step creates $\text{RO}^g(L_a^A, L_b^B) = X_{ab}^g$. The values $(X_{00}^g, X_{01}^g, X_{10}^g, X_{11}^g)$ are intermediate

Algorithm 4. Circuit($e, \mathbf{C}, \ell, \ell'$)

- 1: $\forall g \in [q]$, initialize the random oracle $\text{RO}^g[2\ell, \ell']$
 - 2: initialize the wire label set $W = [W_1, \dots, W_{n+q}]$
 - 3: **for** each circuit input wire A **do**
 - 4: $W_A = (\mathbb{L}_0^A, \mathbb{L}_1^A) \in e$
 - 5: **end for**
 - 6: initialize $F = [], D = []$
 - 7: **for** each gate $g = (f_g, A, B, g)$ in \mathbf{C} in topological order **do**
 - 8: extract input wire labels $\mathbb{L}_0^A, \mathbb{L}_1^A, \mathbb{L}_0^B, \mathbb{L}_1^B \in W$
 - 9: compute $(\mathbb{L}_0^g, \mathbb{L}_1^g, \nabla^g) \leftarrow \text{Gate}(\mathbb{L}_0^A, \mathbb{L}_1^A, \mathbb{L}_0^B, \mathbb{L}_1^B, g, f_g, \ell)$
 - 10: set $F[g] \leftarrow \nabla^g$
 - 11: set $W_g = (\mathbb{L}_0^g, \mathbb{L}_1^g) \in W$
 - 12: **if** g is an output gate **then**
 - 13: $D[g] \leftarrow (\mathbb{L}_0^g, \mathbb{L}_1^g)$
 - 14: **end if**
 - 15: **end for**
 - 16: **Return** (F, D)
-

garbling values termed ‘approximate key’, each ℓ' -bit long, that are the outputs of the random oracle. Note that since ℓ' is an internal length parameter and, as all internal variables are not communicated, it can be arbitrarily long without effecting the communication complexity of the garbling scheme.

Next, the set $(X_{00}^g, X_{01}^g, X_{10}^g, X_{11}^g)$ is used to create a gate garbling ∇^g . The length of ∇^g is $\ell^g \leq \ell'$ and it varies for each garbled gate. Depending on the gate type, Tables 3–4 contain truth-tables that indicate how a single index of ∇^g is set as a function of the bits in the same index in each of $(X_{00}^g, X_{01}^g, X_{10}^g, X_{11}^g)$. These tables are part of the description of the garbling scheme and are fixed prior to running the garbling algorithm.

The garbling ∇^g is generated bit-by-bit until the Hamming weight comes to ℓ , the effective length. The gate garbling ∇^g is also made such that for any intermediate value X_{ab}^g , the output label can be derived as $\nabla^g \circ X_{ab}^g = \mathbb{L}_{f_g(a,b)}^g$ where \circ is an operation that projects the bits in X_{ab}^g over all the positions where ∇^g is set to 1. An essential property that ∇^g satisfies is that on its application with any of the X_{ab}^g , it produces one of two values \mathbb{L}_0^g and \mathbb{L}_1^g that are distributed uniformly at random in $\{0, 1\}^\ell$ and, that too, according to the gate functionality. These, along with the gate garbling ∇^g are the outputs of this subroutine.

Decoding Information. The randomized algorithm $\text{DecodingInfo}(D, \ell) \rightarrow d$ takes the labels set D for the output wires, and returns a sequence d that maps them back to their plain values; see Algorithm 6. This employs a random oracle RO' .

Completing the Garbling Scheme. It remains to describe, for completeness, the working of the input encoding algorithm En , the evaluation algorithm Ev and the output decoding algorithm De . The interfaces and purpose of these are respectively the same as in standard garbling [BHR12]. For brevity we only describe them in algorithmic form in Algorithm 7.

Algorithm 5. $\text{Gate}((L_0^A, L_1^A), (L_0^B, L_1^B), g, f_g, \ell)$

```

1:  $X_{00}^g = \text{RO}^g(L_0^A, L_0^B)$ 
2:  $X_{01}^g = \text{RO}^g(L_0^A, L_1^B)$ 
3:  $X_{10}^g = \text{RO}^g(L_1^A, L_0^B)$ 
4:  $X_{11}^g = \text{RO}^g(L_1^A, L_1^B)$ 
5: initialize  $\nabla^g \leftarrow 0^{\ell'}$  and let  $j = 1$ 
6: repeat
7:   Set  $\text{slice} \leftarrow X_{00}^g[j] || X_{01}^g[j] || X_{10}^g[j] || X_{11}^g[j]$ 
8:   if  $f_g == \text{AND} \wedge \text{slice} \in \{0000, 0001, 1110, 1111\}$  then ▷ See Table 3
9:      $\nabla^g[j] \leftarrow 1$ 
10:  else if  $f_g == \text{XOR} \wedge \text{slice} \in \{0000, 1001, 0110, 1111\}$  then ▷ See Table 4
11:     $\nabla^g[j] \leftarrow 1$ 
12:  end if
13:  increment  $j = j + 1$ 
14: until  $\text{HW}(\nabla^g) = \ell$  or  $j = \ell'$ 
15: if  $\text{HW}(\nabla^g) \neq \ell$  then
16:   ABORT the computation
17: end if
18:  $\ell^g = j$ 
19:  $L_0^g = X_{00}^g \circ \nabla^g$  ▷  $A \circ B = \text{projection of } A[i] \text{ for positions with } B[i] = 1$ 
20: if  $f_g == \text{AND}$  then
21:    $L_1^g = X_{11}^g \circ \nabla^g$ 
22: else if  $f_g == \text{XOR}$  then
23:    $L_1^g = X_{01}^g \circ \nabla^g$ 
24: end if
25: Return  $(L_0^g, L_1^g, \nabla^g)$ 

```

4.2 Motivating Our Scheme

The mainstream literature on garbled circuits has been operating under the gate-by-gate paradigm. Informally, binary gates are individually garbled in topological order. The garbling algorithm samples values for the two labels for each wire (sometimes with additional constraints on their relations) and uses each pair of input labels as a key for encrypting the output label. This is the setting in which [LP09] proves the security of garbling schemes using a primitive that was later termed by [BHR12] a Dual-Key Cipher (DKC). Later, [ZRE15] termed this kind of garbling as ‘linear’. They provided a model for linear garbling and showed that any scheme in their model that simultaneously achieves correctness and privacy requires at least two ciphertexts, thus providing a lower bound on the communication efficiency of such schemes. Our scheme deviates from [ZRE15]’s linear model in several key points.

Approximate Keys. Despite a syntactical similarity, a major difference from prior work is that we do not consider the input labels as keys. Instead, we consider them as an entropy source to an *Approximate-Key-Derivation Function*. This function, modeled as a random oracle and denoted by RO, converts each

Algorithm 6. DecodingInfo(D, ℓ)

```

1: initialize  $RO'[2\ell, 1]$  and  $d = []$ 
2: for output wire  $j \in [m]$  do
3:   extract  $L_0^j, L_1^j \leftarrow D[j]$ 
4:   repeat
5:     sample  $d^j \in_R \{0, 1\}^\ell$ 
6:   until  $RO'(L_0^j, d^j) = 0$  and  $RO'(L_1^j, d^j) = 1$ 
7:    $d[j] \leftarrow d^j$ 
8: end for
9: Return  $d$ 

```

label pair into a uniformly distributed string of length ℓ' . The resulting tuple $t = (X_{00}, X_{01}, X_{10}, X_{11})$ is a set of approximate keys.

Gate Output Label Derivation. The tuple t contains approximate keys in the following sense: $t = (X_{00}, X_{01}, X_{10}, X_{11})$ can be viewed as a $4 \times \ell'$ binary matrix. The garbler scans for each $j \in [\ell']$, indices $\text{slice}_j = (X_{00}[j], X_{01}[j], X_{10}[j], X_{11}[j])$. For an AND gate, the bits in the same column in X_{00}, X_{01} and X_{10} must agree on the same value. When they do, the respective position in L_0^g is set to this value and the respective position in L_1^g is set to the corresponding bit value from X_{11} . Otherwise (i.e., if they do not agree), the value in position j is not included in the construction of the output label.

Table 3 in Appendix A is a truth table according to which the indices slice_j are used to set the j^{th} index of ∇^g when the AND functionality is garbled. Note that the index in ∇^g is set to 1 only in the rows of the table where it holds that $X_{00}[j] = X_{01}[j] = X_{10}[j]$. Further, each index j of ∇^g is set independently, depending on a different slice_j . Each value in this slice is an RO output and so the slice is a uniformly random value in $\{0, 1\}^4$. The right side of Table 3 contains the value in the output label that is a result of projecting the value of X_{ab}^g in the positions where ∇^g contains 1. One can see that L_{00}, L_{01} , and L_{10} always have the same value (and are therefore the same). If anywhere among the ℓ' positions we have $L_{11} \neq (L_{00} = L_{01} = L_{10})$ (i.e., Lines 1 and 14 in Table 3) then, $\{L_0^A, L_1^A\} \times \{L_0^B, L_1^B\} \mapsto \{L_0^g, L_1^g\}$ preserves the structure of a binary AND.

The case for an XOR gate is similar except that the agreement is sought between L_{00} and L_{11} , as well as between L_{01} and L_{10} . Then if at least once in the ℓ' positions $(L_{00} = L_{11}) \neq (L_{01} = L_{10})$, the structure of a binary XOR is preserved. Table 4 in Appendix A is a truth table according to which the indices slice_j are used to set the j^{th} index of ∇^g when the gate functionality is XOR. Note that the index in ∇^g is set to 1 only in the rows of the table where it holds that $X_{00}[j] = X_{11}[j]$ and $X_{01}[j] = X_{10}[j]$.

Both Table 3 and Table 4 are part of the description of the garbling scheme: that is, they are predetermined and remain the same regardless of the function garbled or the randomness used.

Algorithm 7. Algorithms to Evaluate the Garbling

```

1: procedure En( $e, x$ )
2:   initialize  $X = []$ 
3:   for every  $j \in [n]$  do
4:     set  $X[j] = L_{x_j}^j = e_j[x_j]$ 
5:   end for
6:   Return  $X$ 
7: end procedure
8:
9: procedure Ev( $F, X$ )
10:  initialize  $Y = []$ 
11:  for each gate  $g \in [q]$  in a topological order do
12:     $L^A, L^B \leftarrow$  active labels associated with the input wires of gate  $g$ 
13:    extract  $\nabla^g \leftarrow F[g]$  and compute  $L^g \leftarrow \text{RO}(g, L^A, L^B) \circ \nabla^g$ 
14:    if  $g$  is a circuit output wire then
15:       $Y[g] \leftarrow L^g$ 
16:    end if
17:  end for
18:  Return  $Y$ 
19: end procedure
20:
21: procedure De( $Y, d$ )
22:  initialize  $y = []$ 
23:  for  $j \in [m]$  do
24:     $y[j] \leftarrow \text{lsb}(\text{RO}'(Y[j], d^j))$ 
25:  end for
26:  Return  $y$ 
27: end procedure

```

Garbling other gate functionalities. Generalizing the above technique lets us garble an n -input binary gate computing *any* functionality f_g . A gate g with n input wires and one output wire, with each wire holding binary values would have two ℓ -length labels for each wire. For each input wire indexed $i \in [n]$, let these labels be $L_0^i, L_1^i \in \{0, 1\}^\ell$. Garbling such a gate would require a random oracle of the form $\text{RO}^g : \{0, 1\}^{n\ell} \rightarrow \{0, 1\}^{\ell'}$. Let $a = \{a_i\}_{i \in [n]} \in \{0, 1\}^n$ be a possible input value to this gate. The garbling proceeds by first making 2^n calls to the random oracle of the form $\text{RO}^g(\{L_i^{a_i}\}_{i \in [n]}) \rightarrow X_a^g$, for all $a \in \{0, 1\}^n$. Letting $t = \{X_a^g\}_{a \in \{0, 1\}^n}$ be the set of approximate keys, we need that t be partitioned into two sets: $t^0 = \{X_a^g\}_{f_g(a)=0}$ containing all approximate keys that need to be mapped to the gate output label L_0^g , and $t^1 = \{X_a^g\}_{f_g(a)=1}$ with those that are mapped to L_1^g . Each index in $\nabla_{f_g}^g$ is set to 1 only when the values in the slice of t^0 are equal and that of t^1 are equal. This corresponds to 4 possible values of slice: $0^m, 1^m$, the transpose vector of the truth table of f_g , and the complement thereof. In effect, only knowing these 4 combinations suffices for gate garbling.

Let us consider the special case of 2-input binary gates for some functionality f_g . The random oracle is of the form $\text{RO}^g : \{0, 1\}^{2^\ell} \rightarrow \{0, 1\}^{\ell'}$ and there are 4 approximate keys in the tuple $t = (X_{00}^g, X_{01}^g, X_{10}^g, X_{11}^g)$. In the truth-table for setting the bits in $\nabla_{f_g}^g$, out of the $2^4 = 16$ rows, 4 rows would set the bit in $\nabla_{f_g}^g$ to 1. Therefore, when a garbling is generated for a 2-input binary gate, it has the same distribution regardless of the gate functionality. This is the basis of the gate functionality hiding property of the scheme.

No ciphertexts. In the evaluation algorithm Ev of our scheme, given two input labels, the evaluator can obtain from the random oracle exactly one approximate key whereas what they need is the output label. To enable this, the garbler records in ∇ the bit positions from which the output label is derived. We believe ∇ should not be considered a ciphertext since it does not encrypt any labels but instead encodes the relation between an approximate-key and the output label. Furthermore, whereas a ciphertext normally captures the relation between a pair of input labels and the output label, ∇ captures the relation between all input labels and both output labels. Finally, each bit of ∇ is zero with probability $\frac{3}{4}$ and one with probability $\frac{1}{4}$.

4.3 Security

We consider a PPT adversary \mathcal{A} that runs for $t(\kappa)$ time steps for any polynomial $t(\cdot)$ in the security parameter κ . \mathcal{A} has access to the random oracles RO^g and RO' but, owing to its running time, is restricted to making at most $t(\kappa)$ queries overall. Among these, we make a distinction between the set of honest queries H , and adversarial queries Q . Given the challenge (F, X, d) output from Algorithm 1, we term the set of queries made in $\text{Ev}(F, X) = Y$ and $\text{De}(Y, d)$ as the honest queries H . For a circuit \mathbf{C} with q gates and m output wires, this includes q calls to RO , and m calls to RO' . Therefore, $|H| = q + m$ and its contents are determined completely by the challenge. Any other query \mathcal{A} makes is an adversarial query in Q . We only consider sets Q where $|Q| < t(\kappa)$.

Security Game. The security game for Privacy from Definition 2 is an interaction between the adversary \mathcal{A} and the challenger. Let Φ be a leakage function denoting the topology of a circuit. That is, for a circuit \mathbf{C} , $\Phi(\mathbf{C})$ outputs everything except the gate functionality of each gate in the circuit. First \mathcal{A} picks circuits $\mathbf{C}_0, \mathbf{C}_1$ of its choice such that $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$, and two inputs $x^0, x^1 \in \{0, 1\}^n$ such that $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$. Then, $(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1)$ are given to the challenger. On receiving this, the challenger first samples a bit $b \leftarrow \{0, 1\}$ uniformly at random. It then garbles \mathbf{C}_b , creating (F, e, d) . It encodes x^b using e to get X . The challenge (F, X, d) is sent back to \mathcal{A} . Now \mathcal{A} with polynomial running time, and access to the random oracles RO^g , and RO' to make honest queries H and adversarial queries Q , is tasked with guessing b that was used internally by the challenger.

The adversary's view. In the privacy game from Definition 2, \mathcal{A} has in its view $(\mathbf{C}_0, \mathbf{C}_1, x^1, x^0)$ of its own choice, (F, X, d) that it receives as the challenge, the

set of honest queries (and responses) H , and adversarial queries (and responses) Q . We define a function $\mathcal{V}(\cdot)$ that represents the information learnt by \mathcal{A} . For instance, $\mathcal{V}(F, X, d)$ refers to the information \mathcal{A} can deduce from the challenge (F, X, d) . In particular, by $\mathcal{V}(F, X, d, H, Q)$ we denote all the information learnt by the adversary². The advantage Adv of \mathcal{A} can be restated as

$$\text{Adv} = \left| \Pr[\mathcal{A}(\mathcal{V}(F, X, d, H, Q)) = b] - \frac{1}{2} \right|$$

where the probability distribution is taken over the secrets of the challenger (i.e., random choice of $b \leftarrow \{0, 1\}$, and the garbling randomness: $(F, e, d) \leftarrow \text{Gb}(\mathbf{C}_b)$), and the choice of the adversarial query set Q . The adversary needs to distinguish between the cases that the challenger chooses $b = 0$ and $b = 1$.

Theorem 1. *Let $\text{GS} = (\text{Gb}, \text{En}, \text{Ev}, \text{De})$ be a garbling scheme as in Algorithms 2–7. Let κ be a computational security parameter. For every PPT adversary \mathcal{A} with running time $t(\kappa)$ having access to all random oracles $\text{RO} \in (\text{RO}^g, \text{RO}')$ in the Privacy game (Definition 2), \exists a negligible function μ s.t. \mathcal{A} 's advantage is,*

$$\text{Adv}(\kappa) = \left| \Pr[\mathcal{A}^{\text{RO}}(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1, F, X, d) = b] - \frac{1}{2} \right| < \mu(\kappa)$$

Proof Outline. We prove that our garbling scheme preserves privacy against a PPT adversary. The privacy game (Algorithm 1) returns as a challenge (F, X, d) and the adversary \mathcal{A} is tasked with guessing the bit b such that $(F, e, d) \leftarrow \text{Gb}(\mathbf{C}_b)$ and $X = \text{En}(e, x^b)$. Note that the garbling (F, d) in isolation is distributed identically for both \mathbf{C}_0 and \mathbf{C}_1 . This is because the garbling technique creates each gate garbling ∇^g in a way that it is distributed identically regardless of the gate functionality $f_g \in \{\text{AND}, \text{XOR}\}$ and $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$.

We denote by honest queries the RO queries that are necessary for evaluating the garbling F on X . Then our proof follows by proving in Theorem 2 that given the challenge (F, X, d) and the set of honest queries H only, the view of the adversary \mathcal{A} is identically distributed for the cases where $b = 0$ and $b = 1$. In order to show this, we first prove that nothing beyond the *active path* P of the evaluation is revealed from the given information. Next, we show that the active path is identically distributed for both cases.

All queries that are not honest queries are referred to as adversarial queries. When an adversarial query is made, we make a distinction between the case where a response lies in the garbling F , and those that do not. We call the former a ‘Bad Event’. When a ‘Bad Event’ occurs we assume an adversary can detect this, and that it gives it enough information to distinguish for b . Therefore, we bound the probability of a Bad Event for a single query in Theorem 3.

When a query does not lead to a bad event, this implies that its response is irrelevant to the construction of (F, d) . Making such a query does not give

² We omit writing $(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1)$ and other garbling parameters like ℓ and ℓ' for brevity but it is assumed to be always included.

the adversary any advantage over the case considered in Theorem 2. However, it restricts the domain of future queries to the random oracles. As a result, a future query to it may have a higher probability of incurring a bad event. In Theorem 4, we bound the advantage that the adversary would have on making s adversarial queries. This is done by first, calculating the probability that an i^{th} query leads to a bad event given that $i - 1$ previous queries have not triggered a bad event. This probability is an increasing function of i . Next, \mathcal{A} 's advantage is bounded as the complement of the probability that no bad event has occurred in s queries. The probability of no bad event occurring is calculated as the product of the complement of the individual probabilities for each round i that was previously calculated. This result also holds in the case where \mathcal{A} makes $t(\kappa)$ adversarial queries, completing the proof for the main theorem: Theorem 1.

5 Supporting Free-XOR

The garbling scheme in Sect. 4 can be further extended to support free-XOR. The idea is similar to existing free-XOR schemes where the garbler samples a secret global offset $\Delta \in \{0, 1\}^\ell$. For each input wire, the 0-label is sampled uniformly at random and the 1-label is set such that $L_0 \oplus L_1 = \Delta$. The XOR gate is evaluated by setting the output label as the bitwise XOR between the labels of the two input wires. This complies with the XOR gate functionality and maintains the invariant that for the output wire of the XOR gate, $L_0 \oplus L_1 = \Delta$. This gate itself has no garbling representation.

It now remains to show how other gate functionalities like AND are garbled so that the output wire labels maintain the same invariant. This is done by including Δ as one of the constraints, along with $t = (X_{00}, X_{01}, X_{10}, X_{11})$, that is used to create ∇^g . Table 5 in Appendix A indicates the new set of constraints. In this table, the index j in ∇^g is set to 1 only when the indices in $X_{00} = X_{01} = X_{10}$ and when for the desired index j' in Δ , it holds that $X_{00} \oplus X_{11} = \Delta$. Algorithm 8 details the gate garbling algorithm for the AND gate, supporting free-XOR. Note that while the index j is incremented in every iteration, going over all the indices in ∇^g , the index j' is incremented only when one bit in ∇^g is set to 1, so as to move to the next element in Δ . This continues until the ℓ bits in Δ are exhausted.

Note that out of the 32 different ways that $\nabla^g[j]$ can be set in Table 5, only $\frac{1}{8}$ of them sets it to 1 and the rest set the bit to 0. So in order to maintain a Hamming weight of ℓ in ∇^g , its size would become 8ℓ in expectation. Therefore, supporting free-XOR in this scheme incurs the cost of increasing the size of the gate garbling ∇^g by double in expectation.

Although this modification to the scheme for free-XOR compatibility increases the size of the garbled gates, the same security analysis as in the proof in Appendix B, with the exception that the leakage function Φ now not only reveals the topology of the circuit, but also the position of the XOR gates. This leakage function is also invertable and so the extension from indistinguishability based privacy to simulation based privacy still holds.

Algorithm 8. $\text{Gate}((L_0^A, L_1^A), (L_0^B, L_1^B), g, \ell, \Delta)$

- 1: $X_{00}^g = \text{RO}^g(L_0^A, L_0^B), X_{01}^g = \text{RO}^g(L_0^A, L_1^B), X_{10}^g = \text{RO}^g(L_1^A, L_0^B), X_{11}^g = \text{RO}^g(L_1^A, L_1^B)$
 - 2: initialize $\nabla^g \leftarrow 0^{\ell'}$ and let $j = 1, j' = 1$
 - 3: **repeat**
 - 4: **slice** $\leftarrow \Delta[j'] \parallel X_{00}^g[j] \parallel X_{01}^g[j] \parallel X_{10}^g[j] \parallel X_{11}^g[j]$
 - 5: **if** slice $\in \{00000, 10001, 11110, 01111\}$ **then** ▷ See Table 5
 - 6: $\nabla^g[j] \leftarrow 1$ and $j' = j' + 1$
 - 7: **end if**
 - 8: $j = j + 1$
 - 9: **until** $j' == \ell$
 - 10: $L_0^g = X_{00}^g \circ \nabla^g$
 - 11: $L_1^g = X_{11}^g \circ \nabla^g$
 - 12: **Return** (L_0^g, L_1^g, ∇^g)
-

Our security proof follows a ‘Bad Event’ analysis and the advantage that the adversary gains is calculated in terms of the probability of encountering a bad event. Therefore the advantage in the privacy game for this scheme with free-XOR can be calculated as being the same as that for the scheme in Sect. 4. However, there remains a crucial difference between the modified scheme and the original. In the original scheme, we assume for simplicity that if a Bad Event is encountered, the adversary can distinguish and security is violated. However, this may not always be the case. In practice, encountering a bad event would only aid in violating privacy when it is encountered at certain favourable wires or gates, depending on the circuit topology. But, in the modified scheme, encountering a bad event means finding an inactive output label and along with the active value already known, this reveals the value of the global offset Δ . Then all the inactive labels throughout the garbling can be found and the privacy is violated.

Summarizing, in the original scheme, in practice, encountering a Bad Event may not always violate privacy. But, when the scheme is modified for free-XOR, encountering a Bad Event would always violate privacy. Owing to our conservative approach, the proof in Appendix B accounts for the worst case scenario when analyzing the scheme in Sect. 4, and is therefore agnostic to whether free-XOR compatibility was leveraged or not. Therefore both schemes are secure.

Garbling Other Gates. Extending the discussion in Sect. 4.2, a general n -input binary gate computing any functionality f_g can be garbled in a free-XOR compatible way. Here again, a random oracle is used to generate the 2^n approximate keys $t = \{X_a^g\}_{a \in \{0,1\}^n}$. Out of these, let $t^0 = \{X_a^g\}_{f_g(a)=0}$, and $t^1 = \{X_a^g\}_{f_g(a)=1}$. Similar to Table 5, the table for creating $\nabla_{f_g}^g$ contains $m = 2^n + 1$ columns on the left where 2^n columns correspond to the approximate keys and one column is for the global offset Δ . The number of rows in the table would be 2^m out of which only 4 would set ∇^g to 1: the rows where each element in t^0 takes the same value b_0 , elements in t^1 takes value b_1 , and $b_0 \oplus b_1$ equals the value in Δ .

Table 4. For a gate index g and $j \in [\ell']$, this table defines $\nabla_{\oplus}^g[j]$ (where $f_g = \text{XOR}$) as a function in $X_{00}^g[j], X_{01}^g[j], X_{10}^g[j], X_{11}^g[j]$. In addition, the right side demonstrates how $X_{ab}^g[j] \circ \nabla^g[j]$ collapses into only two distinct values $\mathbf{L}_0^g = \mathbf{L}_{00} = \mathbf{L}_{11}$ and $\mathbf{L}_1^g = \mathbf{L}_{01} = \mathbf{L}_{10}$. Each row in the table corresponds to one bit-slice of the values $X_{ab}^g[j]$ for $a, b \in \{0, 1\}$.

	X_{00}^g	X_{01}^g	X_{10}^g	X_{11}^g	∇_{\oplus}^g	\mathbf{L}_{00}	\mathbf{L}_{01}	\mathbf{L}_{10}	\mathbf{L}_{11}
0	0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	-	-	-	-
2	0	0	1	0	0	-	-	-	-
3	0	0	1	1	0	-	-	-	-
4	0	1	0	0	0	-	-	-	-
5	0	1	0	1	0	-	-	-	-
6	0	1	1	0	1	0	1	1	0
7	0	1	1	1	0	-	-	-	-
8	1	0	0	0	0	-	-	-	-
9	1	0	0	1	1	1	0	0	1
10	1	0	1	0	0	-	-	-	-
11	1	0	1	1	0	-	-	-	-
12	1	1	0	0	0	-	-	-	-
13	1	1	0	1	0	-	-	-	-
14	1	1	1	0	0	-	-	-	-
15	1	1	1	1	1	1	1	1	1

ℓ positions with 1. From Table 3, 4 it is evident that a position j in ∇^g is set to 1 with probability $\frac{1}{4}$ over a random choice of $(X_{00}[j], X_{01}[j], X_{10}[j], X_{11}[j]) \in \{0, 1\}^4$. As these bits originate from random oracle outputs, they are indeed distributed uniformly at random. Therefore, ℓ' needs to be set such that the probability of ∇^g having Hamming weight $< \ell$ is negligible in κ . Let us now examine this probability for $\ell' = 8\kappa$.

For a gate g , let H be a random variable that denotes the Hamming weight of ∇^g derived from a random $t = (X_{00}, X_{01}, X_{10}, X_{11})$ where each $X_{ab} \in \{0, 1\}^{\ell'}$.

$$H \sim \text{Binomial}(\ell', \frac{1}{4}) = \text{Binomial}(8\kappa, \frac{1}{4})$$

This random variable has a mean $\mu = np = \frac{8\kappa}{4} = 2\kappa$ and variance $\sigma^2 = npq = 1.5\kappa$. Then, using Hoeffding's inequality for Binomial Distributions,

$$\begin{aligned} \Pr[H < \kappa] &\leq e^{-2n(p - \frac{\kappa}{n})^2} \\ &= e^{-16\kappa(\frac{1}{4} - \frac{\kappa}{8\kappa})^2} \\ &= e^{-\frac{\kappa}{4}} \end{aligned}$$

which is negligible in κ .

Table 5. For a gate index g , $j \in [\ell']$ and $j' \in [\ell]$, this table defines $\nabla_{\wedge}^g[j]$ (where $f_g = \text{AND}$) as a function of $X_{00}^g[j], X_{01}^g[j], X_{10}^g[j], X_{11}^g[j]$ and $\Delta[j']$. In addition, the right side demonstrates how combining $X_{ab}^g[j] \circ \nabla^g[j]$ collapses into only two distinct values $L_0^g = L_{00} = L_{01} = L_{10}$ and $L_1^g = L_{11}$ such that $L_0^g \oplus L_1^g = \Delta$. Each row in the table corresponds to one bit-slice of the values $X_{ab}^g[j]$ for $a, b \in \{0, 1\}$.

	Δ	X_{00}^g	X_{01}^g	X_{10}^g	X_{11}^g	∇_{\wedge}^g	L_{00}	L_{01}	L_{10}	L_{11}
0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	1	0	-	-	-	-
2	0	0	0	1	0	0	-	-	-	-
3	0	0	0	1	1	0	-	-	-	-
4	0	0	1	0	0	0	-	-	-	-
5	0	0	1	0	1	0	-	-	-	-
6	0	0	1	1	0	0	-	-	-	-
7	0	0	1	1	1	0	-	-	-	-
8	0	1	0	0	0	0	-	-	-	-
9	0	1	0	0	1	0	-	-	-	-
10	0	1	0	1	0	0	-	-	-	-
11	0	1	0	1	1	0	-	-	-	-
12	0	1	1	0	0	0	-	-	-	-
13	0	1	1	0	1	0	-	-	-	-
14	0	1	1	1	0	0	-	-	-	-
15	0	1	1	1	1	1	1	1	1	1
16	1	0	0	0	0	0	-	-	-	-
17	1	0	0	0	1	1	0	0	0	1
18	1	0	0	1	0	0	-	-	-	-
19	1	0	0	1	1	0	-	-	-	-
20	1	0	1	0	0	0	-	-	-	-
21	1	0	1	0	1	0	-	-	-	-
22	1	0	1	1	0	0	-	-	-	-
23	1	0	1	1	1	0	-	-	-	-
24	1	1	0	0	0	0	-	-	-	-
25	1	1	0	0	1	0	-	-	-	-
26	1	1	0	1	0	0	-	-	-	-
27	1	1	0	1	1	0	-	-	-	-
28	1	1	1	0	0	0	-	-	-	-
29	1	1	1	0	1	0	-	-	-	-
30	1	1	1	1	0	1	1	1	1	0
31	1	1	1	1	1	0	-	-	-	-

B Proof of Theorem 1

Before stating the proof itself, we define certain terms used within our proof.

B.1 Proof Setup

In the security game, the adversary’s goal given (F, X, d) is to distinguish whether $(F, d) \leftarrow \text{Gb}(C_0)$ and $X = \text{En}(e, x^0)$, or $(F, d) \leftarrow \text{Gb}(C_1)$ and $X = \text{En}(e, x^1)$. Going forward, for a gate g , we denote by L^A and L^B the active input labels, and by L^g the active output label. These values are revealed during the evaluation of F on X . We show in our proof that the knowledge of these *active values only* gives the adversary \mathcal{A} zero advantage in distinguishing b .

We denote by L^{A*} and L^{B*} the inactive input labels, and by L^{g*} the inactive output label. We term as a “Bad Event”, the case where an adversary \mathcal{A} learns an inactive label for any wire in F . These reveal additional information about the circuit and potentially its correlation to X , leading the adversary to gain advantage in distinguishing in the privacy game. For simplicity of analysis, we assume that when a random oracle query leads to a ‘Bad Event’, privacy is already violated, without needing further work/queries from the adversary. For any wire indexed i , we denote by $L^{i'}$ a *candidate* for an inactive label. Such a candidate is queried to the random oracle in order to learn whether it is the inactive label or not. We bound the probability of a “Bad Event” by computing a bound on the number of such possible queries. We then argue that for each query to $\text{RO} \in (\text{RO}^g, \text{RO}')$, the probability of encountering a bad event is negligible.

We denote by Q the set of *adversarial queries and responses*. Setting $|Q| = s$,

$$Q = \left\{ [g_i, q_i, r_i] \right\}_{i \in [s]}$$

where g_i is the gate index for when RO is queried, q_i is the value input during the i^{th} query, and r_i is its respective response.

We denote by H , the set of *honest queries and responses*. Given the challenge (F, X, d) , this is the set of queries to RO^g, RO' that are made within $\text{Ev}(F, X) = Y$ and $\text{De}(Y, d) = y$. The set H has the following form:

$$H = \left\{ \begin{aligned} & \{ [g, (L^{A_g}, L^{B_g}), X^{AB_g}] \}_{g \in [q]} \\ & \{ [-, (Y[j], d^j), y_j] \}_{j \in [m]} \end{aligned} \right.$$

We denote by P the corresponding *active path* in F . P contains all the values revealed when evaluating the garbling (F, d) on X . All the elements in P can be derived from the elements in H . P has the form:

$$P = \left\{ \left\{ L^{A_g}, L^{B_g}, X^{AB_g}, L^g \right\}_{g \in [q]} \cup \{ Y[j], d^j, y_j \}_{j \in [m]} \right\}$$

Note that all the labels L^w for each wire $w \in [n + q]$ are of length ℓ -bits (Table 2). This is also the length of the output labels $Y[j]$ and decoding labels

d^j for each output wire $j \in [m]$. For each gate $g \in [q]$, the values X^{AB_g} has length ℓ' -bits. This is an upper bound on the length of the gate garbling ∇^g . Each ∇^g has Hamming weight ℓ . This Hamming weight is the *effective key length* and the indices in ∇^g containing 1 are termed as the *effective key positions*.

Definition 3 (Bad Event 1). For a gate g with a garbling ∇^g , let \mathcal{L}^B be the set of candidate inactive labels for input wire B . Let \mathcal{L}^{g*} be the inactive output label and \mathcal{L}^g be the active output label. ‘Bad Event 1’ occurs when for $\mathcal{L}^{b'} \in \mathcal{L}^B$ that is queried by the adversary to RO , it holds that,

$$\text{RO}^g(\mathcal{L}^A, \mathcal{L}^{b'}) \circ \nabla^g \in \{\mathcal{L}^g, \mathcal{L}^{g*}\}$$

For simplicity, we treat the test for whether a candidate output label $\mathcal{L}^{g'}$ is the inactive label \mathcal{L}^{g*} as requiring zero additional calls after the call to RO^g for Bad Event 1 (Definition 3).

Lemma 1. In the same setting as in Definition 3, let $B_{\mathcal{L}^B} \subseteq \mathcal{L}^B$ be the set of candidates leading to ‘Bad Event 1’, $\mathcal{L}^{b'}$ be the candidate queried in the i -th query, and $\mathcal{L}_i \subseteq \mathcal{L}^B$ the set consisting of the previous $i - 1$ queried candidates. For effective key length ℓ of ∇^g it holds that,

$$\Pr[\text{RO}^g(\mathcal{L}^A, \mathcal{L}^{b'}) \circ \nabla^g \in \{\mathcal{L}^g, \mathcal{L}^{g*}\} | B_{\mathcal{L}^B} \cap \mathcal{L}_i = \emptyset] \leq \frac{1}{2^{\ell-i}} + 2^{-\ell+1}$$

i.e., the probability that the i -th query triggers ‘Bad Event 1’ is upper bounded by $\frac{1}{2^{\ell-i-1}} + 2^{-\ell+1}$ as long as none of the previous queries triggered the same.

Proof: Let $\mathcal{S} = \mathcal{L}^B - \mathcal{L}_i$ be the set of candidate input labels that have not yet been queried. Note that the size of the set $\mathcal{S} \geq 2^\ell - i - 1$. Let E be the event that $B_{\mathcal{L}^B} \cap \mathcal{L}_i = \emptyset \cap \mathcal{L}^{b'} \notin \mathcal{L}_i$, that is, a new label is being queried and none of the previous $i - 1$ queries have triggered a Bad Event. We calculate the probability of ‘Bad Event 1’ by considering two cases. One case is when the inactive input label is chosen: $\mathcal{L}^{b'} = \mathcal{L}^{B*} \in \mathcal{S}$. Querying on this yields one of \mathcal{L}^{g*} or \mathcal{L}^g (according to the gate functionality) with probability 1. The other case is when any other candidate $\mathcal{L}_i^{b'} \in \mathcal{S}$ is picked. Since the output of RO^g is a truly random string in $\{0, 1\}^\ell$, it can yield \mathcal{L}^{g*} or \mathcal{L}^g with probability $\frac{2}{2^\ell}$. Therefore,

$$\begin{aligned} & \Pr[\text{RO}^g(\mathcal{L}^A, \mathcal{L}^{b'}) \circ \nabla^g \in \{\mathcal{L}^g, \mathcal{L}^{g*}\} | E] \\ &= \Pr[\text{RO}^g(\mathcal{L}^A, \mathcal{L}^{b'}) \circ \nabla^g \in \{\mathcal{L}^g, \mathcal{L}^{g*}\} | E, \mathcal{L}^{b'} = \mathcal{L}^{B*}] \cdot \Pr[\mathcal{L}^{b'} = \mathcal{L}^{B*} | E] \\ &+ \Pr[\text{RO}^g(\mathcal{L}^A, \mathcal{L}^{b'}) \circ \nabla^g \in \{\mathcal{L}^g, \mathcal{L}^{g*}\} | E, \mathcal{L}^{b'} \neq \mathcal{L}^{B*}] \cdot \Pr[\mathcal{L}^{b'} \neq \mathcal{L}^{B*} | E] \\ &= 1 \cdot \frac{1}{2^{\ell-i-1}} + \frac{2}{2^\ell} \cdot \frac{2^\ell - i - 2}{2^\ell - i - 1} \\ &\approx \frac{1}{2^{\ell-i-1}} + 2^{-\ell+1} \end{aligned} \quad \square$$

Symmetrically, we define Bad Event 2 and 3 as follows:

Definition 4 (Bad Event 2). For a gate g with a garbling ∇^g , let \mathcal{L}^A be the set of candidate inactive labels for input wire A . Let L^{g*} be the inactive output label and L^g be the active output label. ‘Bad Event 2’ occurs when for $\mathsf{L}^{a'} \in \mathcal{L}^A$, it holds that,

$$\mathsf{RO}^g(\mathsf{L}^{a'}, \mathsf{L}^B) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\}$$

Definition 5 (Bad Event 3). For a gate g with a garbling ∇^g , let \mathcal{L}^B and \mathcal{L}^A be the set of candidate inactive labels for input wire B and A respectively. Let L^{g*} be the inactive output label and L^g be the active output label. ‘Bad Event 3’ occurs when for $\mathsf{L}^{b'} \in \mathcal{L}^B$ and $\mathsf{L}^{a'} \in \mathcal{L}^A$, it holds that,

$$\mathsf{RO}^g(\mathsf{L}^{a'}, \mathsf{L}^{b'}) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\}$$

Corollary 1. In the same setting as Definition 4, let $B_{\mathcal{L}^A} \subseteq \mathcal{L}^A$ the set of candidates leading to ‘Bad Event 2’, $\mathsf{L}^{a'}$ be the candidate queried in the i -th query, and $\mathcal{L}_i \subseteq \mathcal{L}^A$ the set consisting of the previous $i - 1$ queried candidates. For effective key length ℓ of ∇^g it holds that,

$$\Pr[\mathsf{RO}^g(\mathsf{L}^{a'}, \mathsf{L}^B) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\} | B_{\mathcal{L}^A} \cap \mathcal{L}_i = \emptyset] \leq \frac{1}{2^{\ell-i}-1} + 2^{-\ell+1}$$

i.e., the probability that the i -th query triggers ‘Bad Event 2’ is upper bounded by $\frac{1}{2^{\ell-i}} + 2^{-\ell+1}$ as long as none of the previous queries triggered the same.

Corollary 2. In the same setting as Definition 5, let $B_{\mathcal{L}^A, \mathcal{L}^B} \subseteq \mathcal{L}^A \times \mathcal{L}^B$ be the ordered set of candidates leading to ‘Bad Event 3’, $\mathsf{L}^{b'}$ and $\mathsf{L}^{a'}$ be the candidate queried in the i -th query, and $\mathcal{L}_i \subseteq \mathcal{L}^A \times \mathcal{L}^B$ be the set consisting of the previous $i - 1$ queries. For effective key length ℓ of ∇^g it holds that,

$$\Pr[\mathsf{RO}^g(\mathsf{L}^{a'}, \mathsf{L}^{b'}) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\} | B_{\mathcal{L}^A, \mathcal{L}^B} \cap \mathcal{L}_i = \emptyset] \leq \frac{1}{2^{\ell-i}-1} + 2^{-\ell+1}$$

i.e., the probability that the i -th query triggers ‘Bad Event 3’ is upper bounded by $\frac{1}{2^{\ell-i}} + 2^{-\ell+1}$ as long as none of the previous queries triggered the same.

B.2 The Complete Proof

Theorem 2 (Honest-but-Curious Adversarial Behaviour). Let \mathcal{A} be a PPT adversary. In the privacy game as in Algorithm 1, given $(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1)$ of \mathcal{A} ’s choice such that $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$ and $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$, the challenge (F, X, d) , and H , the set of honest queries only, it holds that,

$$\begin{aligned} & \Pr[F, d \leftarrow \mathsf{Gb}(\mathbf{C}_0), X = \mathsf{En}(e, x^0) | \mathcal{V}(F, X, d, H)] \\ &= \Pr[F, d \leftarrow \mathsf{Gb}(\mathbf{C}_1), X = \mathsf{En}(e, x^1) | \mathcal{V}(F, X, d, H)] \end{aligned}$$

Proof: Before proving the above theorem, consider the following lemmas:

Lemma 2 (Honest Queries reveal only the Active Path). *Let (F, X, d) be the challenge that is output from Algorithm 1. Let H be the set of honest queries and let P be the active path. Then,*

$$\mathcal{V}(F, X, d, H) = P$$

Proof: The proof follows in two steps. First we need to show that P can indeed be derived from $\mathcal{V}(F, X, d, H)$. That is,

$$P \subseteq \mathcal{V}(F, X, d, H)$$

This holds by construction. The active path P can be derived from $\mathcal{V}(F, X, d, H)$ since all of its elements can be determined from H . Recall, H is the set of honest queries to the random oracles that are necessary for computing $Y = \text{Ev}(F, X)$ and $y = \text{De}(Y, d)$ from the challenge. By definition, it has the form,

$$H = \left\{ \begin{array}{l} \{[g, (\mathbb{L}^{A_g}, \mathbb{L}^{B_g}), X^{AB_g}]\}_{g \in [q]} \\ \{[-, (Y[j], d^j), y_j]\}_{j \in [m]} \end{array} \right.$$

So (F, X, d, H) does indeed complete all the information in the active path:

$$P = \left\{ \{ \mathbb{L}^{A_g}, \mathbb{L}^{B_g}, X^{AB_g}, \mathbb{L}^g \}_{g \in [q]} \cup \{ Y[j], d^j, y_j \}_{j \in [m]} \right\}$$

In order to complete the proof of the theorem, it remains to show that nothing beyond P is revealed from $\mathcal{V}(F, X, d, H)$. That is,

$$P \supseteq \mathcal{V}(F, X, d, H)$$

We show that P alone can be used to recreate the tuple (F, X, d, H) . First, note that X contains the set of active labels for all circuit input wires. This is contained within P . The set $d = \{d^j\}_{j \in [m]}$ is the decoding information, also contained within P . H can also be determined by P . For each gate g , the elements $(\mathbb{L}^{A_g}, \mathbb{L}^{B_g}, X^{AB_g}) \in P$ are the query and response for RO^g . The set $\{Y[j], d^j, y_j\}_{j \in [m]}$ is the set of RO' query and responses in H . Finally, F is a set of gate garblings, ∇^g . For each $g \in [q]$, this can be derived from examining X^{AB_g} and \mathbb{L}^g : ∇^g is set to 1 for only those positions in X^{AB_g} whose projection gives \mathbb{L}^g . This completes the proof. \square

Lemma 3 (Active Paths are Identically Distributed). *For the garbling $(F_0, d_0, e) \leftarrow \text{Gb}(\mathbf{C}_0)$, let $X_0 = \text{En}(e, x^0)$ and let P_0 and H_0 be the corresponding active path and honest queries set. Similarly, For the garbling $(F_1, d_1, e) \leftarrow \text{Gb}(\mathbf{C}_1)$, let $X_1 = \text{En}(e, x^1)$ and let P_1 and H_1 be the active path and honest queries set. Then if $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$ and $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$, it holds that,*

$$\{F_0, d_0, X_0, P_0, H_0\} \equiv \{F_1, d_1, X_1, P_1, H_1\}$$

Proof: The proof for this considers the distribution $A_0 = \{F_0, d_0, X_0, P_0, H_0\}$ that is derived using \mathbf{C}_0 and x^0 , and the distribution $A_1 = \{F_1, d_1, X_1, P_1, H_1\}$ that is derived using \mathbf{C}_1 and x^1 . Let us examine these distributions:

- In both distributions, the garbling $(F_0, d_0) \in A_0$ and $(F_1, d_1) \in A_1$ are distributed the same way. The garbling F_0, d_0 are a garbling of \mathbf{C}_0 , and F_1, d_1 are a garbling of \mathbf{C}_1 using the garbling scheme in Algorithms 2-6. It holds that their topology, $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$. The garbling produced does not reveal any information beyond Φ . This is because the gate garbling ∇^g is distributed the same way regardless of the functionality $f_g \in \{\text{AND}, \text{XOR}\}$ due to the nature of Algorithm 5 and Table 3, 4.
- Considering the complete challenge $(F_0, d_0, X_0) \in A_0$ and $(F_1, d_1, X_1) \in A_1$, note that the active input labels sets contain labels that are sampled independently and uniformly at random from $\{0, 1\}^\ell$. These distributions, without making any random oracle queries, is also identically distributed since X and F, d are independent when no RO queries are made.
- On evaluating the challenges, note that $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$ and so the distributions cannot be distinguished on the basis of the output of the evaluation. The honest queries in the set H_0 are determined by (F_0, X_0, d_0) . The distribution of these queries is identical to that in H_1 that are determined by (F_1, X_1, d_1) . This is because the probability that the random oracle query responses are distributed as in H_0 is the same as the probability of it being as in H_1 . Therefore $(F_0, d_0, X_0, H_0) \in A_0$ and $(F_1, d_1, X_1, H_1) \in A_1$ are identically distributed.
- The active paths P_0 and P_1 are determined completely by H_0 and H_1 .

Therefore,

$$\{F_0, d_0, X_0, P_0, H_0\} \equiv \{F_1, d_1, X_1, P_1, H_1\}$$

From Lemma 2, given (F, X, d) and the honest queries H only, nothing beyond the active path P is revealed. Lemma 3 shows that the active paths for any \mathbf{C}_0, x^0 and \mathbf{C}_1, x^1 is identically distributed. Therefore, the theorem follows.

Theorem 3 shows a bound on the advantage from a single adversarial query.

Theorem 3 (Advantage of a single malicious query). *Let \mathcal{A} be a PPT adversary and ℓ be the effective key length. Given the challenge (F, X, d) as in Algorithm 1, \mathcal{A} 's advantage on a single adversarial query is bounded by,*

$$\text{Adv}_{|\mathcal{Q}|=1} \leq 2^{-\ell} + \frac{1}{2^\ell - 2}$$

Proof: From Lemma 1 and Corollary 1 and 2, we can conclude that when the adversary \mathcal{A} makes one adversarial query, it can encounter at most one of the 3 ‘Bad Events’. It can also only gain advantage if the query it makes corresponds to a ‘Bad Event’. Therefore, \mathcal{A} 's advantage on a single adversarial query is,

$$\text{Adv}_{|Q|=1} \leq \Pr[\text{Bad Event}] \leq \max_{j \in [3]} (\Pr[\text{Bad Event } j]) \leq \frac{1}{2^\ell - 2} + 2^{-\ell}$$

□

Theorem 4 extends the result above to provide a bound on the advantage gained from multiple adversarial queries.

Theorem 4 (Advantage in multiple malicious queries). *Let \mathcal{A} be a PPT adversary and ℓ be the effective key length. Given (F, X, d) as in Algorithm 1, the honest queries set H , and adversarial queries Q s.t. $|Q| = s$, \mathcal{A} 's advantage is,*

$$\text{Adv}_{|Q|=s} < \frac{s}{2^\ell - 2}$$

Proof: In order to prove the above theorem, note that a query made by an adversary \mathcal{A} can be broadly classified under one of the following categories:

1. An *Honest Query* where the query and the response for the random oracle lies on the active path of the garbling in the challenge (F, X, d) . Theorem 2 shows that given all the queries H in the active path, \mathcal{A} 's advantage is 0.
2. An *Adversarial Query yielding a 'Bad Event'* is a query other than an Honest Query for which the response of the random oracle lies within the garbling in the challenge. This may reveal information about the garbling beyond the active path. On such an event, without loss of generality, we consider privacy as violated. Our proof builds towards bounding the probability of this event.
3. An *Adversarial Query not yielding a 'Bad Event'* is a random oracle query and response that can evidently not be involved in the construction of the challenge garbling. Making queries to the RO that yield such responses do not help identify the inactive path and therefore give no advantage. That is, given the honest-query-set H , and adversarial queries that do not lead to a 'Bad Event', this will at most help narrow down the domain of the RO. This helps increase the probability of eventually encountering a 'Bad Event'. However, until the 'Bad Event' is encountered this gives \mathcal{A} no advantage over possessing H .

Let q_i be the event that the i^{th} adversarial query takes place given that all $i - 1$ queries before it have not lead to any bad event. We have from Lemma 1, and Corollary 1 and 2 that each of the 'Bad Events' 1, 2 and 3 are bounded as,

$$\Pr[\text{Bad Event} \in \{1, 2, 3\} | q_i] \approx \frac{1}{2^\ell - i - 1} + 2^{-\ell}$$

Note that the probability of the 'Bad Event' increases with the increase in the number of queries and in each query, the probability of encountering any 'Bad Event' at all is calculated as the maximum of these above probabilities. Let us now compute, the probability that a 'Bad Event' is encountered given $|Q| = s$ adversarial queries to the same random oracle:

$$\begin{aligned}
 \Pr[\text{Bad Event} \mid |Q| = s] &= 1 - \Pr[\neg \text{Bad Event} \mid |Q| = s] \\
 &= 1 - \prod_{i=1}^s 1 - \Pr[\text{Bad Event} \mid q_i] \\
 &< 1 - \prod_{i=1}^s \left(1 - \frac{1}{2^{\ell - i - 1}} - 2^{-\ell} \right) \\
 &\approx 1 - \prod_{i=1}^s \left(\frac{2^{\ell} - i - 2}{2^{\ell} - i - 1} \right) = \frac{s}{2^{\ell} - 2}
 \end{aligned}$$

We have seen in the proof for Theorem 3 that one adversarial query can trigger at most 1 ‘Bad Event’ and the adversary \mathcal{A} ’s advantage is bounded by the probability of a ‘Bad Event’ occurring. Given an adversarial query, if the response leads to a ‘Bad Event’, we assume that privacy is violated. If it does not, the views of the adversary are still identical. We therefore need to calculate the probability of at least one ‘Bad Event’ among $|Q| = s$ adversarial queries.

The above is a bound on the probability of a ‘Bad Event’ on a particular random oracle $\text{RO} \in (\text{RO}^g, \text{RO}')$. It remains to extend this result to the case where adversarial queries were made to different random oracles across different gate garblings in the circuit. Note that each random oracle used in the construction is independent. So the result of queries to one random oracle do not affect the result of making (even the same) queries to a different random oracle, except for possibly reusing the query space as a result of seeing a query output without triggering a bad event. So the above is an upper bound that also extends to the case where not all of the previous $i - 1$ queries have been made to the same random oracle since all those cases are bounded by this case. Hence it follows again that when $|Q| = s$, \mathcal{A} ’s advantage is bounded by:

$$\text{Adv}_{|Q|=s} < \frac{s}{2^{\ell} - 2}$$

Summing up, the proof of our final theorem follows. □

Theorem 1 (Overall advantage of a malicious adversary - Restated).

Let $\text{GS} = (\text{Gb}, \text{En}, \text{Ev}, \text{De})$ be a garbling scheme as in Algorithms 2–7. Let κ be a computational security parameter. For every PPT adversary \mathcal{A} with running time $t(\kappa)$ having access to all random oracles $\text{RO} \in (\text{RO}^g, \text{RO}')$, participating in the Privacy game (Definition 2), \exists negligible function μ s.t. \mathcal{A} ’s advantage is,

$$\text{Adv}(\kappa) = \left| \Pr[\mathcal{A}^{\text{RO}}(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1, F, X, d) = b] - \frac{1}{2} \right| < \mu(\kappa)$$

Proof: For a PPT adversary \mathcal{A} running for $t(\kappa)$ time steps, $|Q| \leq t(\kappa)$. Setting $\ell = \kappa$, we have from Theorem 4,

$$\text{Adv}_{|Q|=t(\kappa)} < \frac{t(\kappa)}{2^{\kappa} - 2} \quad \square$$

References

- [AAC+21] Acharya, A., Ashur, T., Cohen, E., Hazay, C., Yanai, A.: A new approach to garbled circuits. *Cryptology ePrint Archive*, Paper 2021/739 (2021). <https://eprint.iacr.org/2021/739>
- [App13] Applebaum, B.: Bootstrapping obfuscators via fast pseudorandom functions. *IACR Cryptol. ePrint Arch.*, p. 699 (2013)
- [App17] Applebaum, B.: Garbled circuits as randomized encodings of functions: a primer. In: *Tutorials on the Foundations of Cryptography*, pp. 1–44 (2017)
- [BHKR13] Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In *IEEE SP*, pp. 478–492 (2013)
- [BHR12] Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: *ACM CCS*, pp. 784–796 (2012)
- [BLO16] Ben-Efraim, A., Lindell, Y., Omri, E.: Optimizing semi-honest secure multiparty computation for the internet. In: *ACM SIGSAC*, pp. 578–590 (2016)
- [BMR90] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: *ACM*, pp. 503–513 (1990)
- [FNO15] Frederiksen, T.K., Nielsen, J.B., Orlandi, C.: Privacy-free garbled circuits with applications to efficient zero-knowledge. In: *EUROCRYPT*, pp. 191–219 (2015)
- [GGP10] Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: outsourcing computation to untrusted workers. In: *CRYPTO*, pp. 465–482 (2010)
- [GKP+13] Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: *STOC*, pp. 555–564. *ACM* (2013)
- [GKPS18] Ganesh, C., Kondi, Y., Patra, A., Sarkar, P.: Efficient adaptively secure zero-knowledge from garbled circuits. In: *PKC*, pp. 499–529 (2018)
- [HK20] Heath, D., Kolesnikov, V.: Stacked garbling - garbled circuit proportional to longest execution path. In: *CRYPTO*, pp. 763–792 (2020)
- [IK00] Ishai, Y., Kushilevitz, E.: Randomizing polynomials: a new representation with applications to round-efficient secure computation. In: *FOCS*, pp. 294–304 (2000)
- [Ish13] Ishai, Y.: Randomization techniques for secure computation. In: *Secure Multi-Party Computation*, volume 10 of *Cryptology and Information Security Series*, pp. 222–248 (2013)
- [KKS16] Kempka, C., Kikuchi, R., Suzuki, K.: How to circumvent the two-ciphertext lower bound for linear garbling schemes. In: *ASIACRYPT*, pp. 967–997 (2016)
- [KL14] Katz, J., Lindell, Y.: *Introduction to Modern Cryptography*, Second Edition (2014)
- [KMR14] Kolesnikov, V., Mohassel, P., Rosulek, M.: Flexor: flexible garbling for XOR gates that beats free-xor. In: *CRYPTO*, pp. 440–457 (2014)
- [KS08] Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: *ICALP*, pp. 486–498 (2008)
- [LP09] Lindell, Y., Pinkas, B.: A proof of security of Yao’s protocol for two-party computation. *J. Cryptol.* **22**(2), 161–188 (2009)
- [NPS99] Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: *ACM-EC*, pp. 129–139 (1999)

- [PSSW09] Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: ASIACRYPT, pp. 250–267 (2009)
- [RR21] Rosulek, M., Roy, L.: Three halves make a whole? beating the half-gates lower bound for garbled circuits. In: CRYPTO, pp. 94–124 (2021)
- [Yao86] Chi-Chih Yao, A.: How to generate and exchange secrets (extended abstract). In: FoCS, pp. 162–167 (1986)
- [ZRE15] Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: EUROCRYPT, pp. 220–250 (2015)

Blockchain



Mt. Random: Multi-tiered Randomness Beacons

Ignacio Cascudo¹, Bernardo David^{2(✉)}, Omer Shlomovits³,
and Denis Varlakov³

¹ IMDEA Software Institute, Madrid, Spain

`ignacio.cascudo@imdea.org`

² IT University of Copenhagen, Copenhagen, Denmark

`bernardo@bmdavid.com`

³ Tel Aviv-Yafo, Israel

`denis@varlakov.me`

Abstract. Many decentralized applications require a common source of randomness that cannot be biased or predicted by any single party. Randomness beacons provide such a functionality, allowing parties to periodically obtain fresh random outputs and verify that they are computed correctly. In this work, we propose Mt. Random, a multi-tiered randomness beacon that combines Publicly Verifiable Secret Sharing (PVSS) and (Threshold) Verifiable Random Function (VRF) techniques in order to provide efficiency/randomness quality trade-offs with security under the standard DDH assumption (in the random oracle model) using only a bulletin board as setup (a requirement for the vast majority of beacons). Each tier provides a constant stream of random outputs offering progressive efficiency vs. quality trade-offs: true uniform randomness is refreshed less frequently than pseudorandomness, which in turn is refreshed less frequently than (bounded) biased randomness. This wide span of efficiency/quality allows for applications to consume random outputs from an optimal point in this trade-off spectrum. In order to achieve these results, we construct two new building blocks of independent interest: GULL, a PVSS-based beacon that preprocesses a large batch of random outputs but allows for gradual release of smaller “sub-batches”, which is a first in the literature of randomness beacons; and a *publicly verifiable* and *unbiasable* protocol for Distributed Key Generation protocol (DKG), which is significantly more efficient than most of previous DKGs secure under standard assumptions and closely matches the efficiency of the currently most efficient *biasable* DKG protocol. We showcase the

Ignacio Cascudo was supported by the Spanish Government under the project SecuRing (ref. PID2019-110873RJ-I00/MCIN/AEI/10.13039/501100011033), by the Madrid Government as part of the program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union, and by a research grant from Nomadic Labs and the Tezos Foundation.

Bernardo David was supported by the Concordium Foundation and by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA²C), 9131-00075B (PUMA) and 0165-00079B.

efficiency of our novel building blocks and of the Mt. Random beacon via benchmarks made with a prototype implementation.

1 Introduction

Randomness is essential for constructing provably secure cryptographic primitives and protocols. For several applications, it does not suffice that parties simply have a local source of randomness, but we require instead a randomness beacon that can periodically provide the same fresh unbiased and unpredictable random values to all parties. This is particularly important in decentralized applications such as Proof-of-stake blockchains (*e.g.* [12, 16, 26]), sharding protocols [37] and smart contracts that need randomness. In such settings, it is desirable to implement a random beacon as a protocol among the mutually distrustful participants of the corresponding system without trusting any single party. Moreover, such protocols must have guaranteed output delivery, publicly verifiable outputs.

Notice that a simple coin tossing protocol where parties commit to local randomness and then output the sum of the opened values is not sufficient, as parties can bias the output with a selective abort strategy, where they open or not their commitments depending on their view so far. Hence, several alternatives for constructing randomness beacons have been proposed based on publicly verifiable secret sharing (PVSS) [4, 8, 9, 26, 33, 35], verifiable random functions (VRF) [12, 15, 16, 20, 23, 36], verifiable delay functions (VDF) [2, 3, 5, 32, 38] and homomorphic encryption [13]. Moreover, achieving fairness against rational adversaries via financial punishments has been proposed by the RANDAO project [30].

Constructions using plain VRFs require very little computation and communication, but are open to the aforementioned selective abort bias. Since they rely on the computation of a VRF by a party who has a certain secret key, an adversarial party can always bias the final output by choosing whether to reveal or not their own VRF output under its secret key (see [16]). Threshold (also called distributed) VRFs, or TVRFs, solve this by always allowing a large enough set of parties (*e.g.* a majority of parties) to compute the verifiable random function, after a setup that consists on a distributed key generation protocol. However, current TVRF-based random beacons protocols simply apply the TVRF to the previous output of the beacon, defining the new beacon output as some fixed function of the new TVRF output. This approach requires a fixed initial seed to which the TVRF is applied in the first round, and since the entropy of such seed is of course finite, the unpredictability guarantees of the process will on the long run necessarily deteriorate. To the best of our knowledge there is no analysis of how exactly this plays out.

Finally, PVSS-based beacons such as SCRAPE [8] and ALBATROSS [9] enhance the commit-and-open strategy mentioned above by having parties commit to their inputs via PVSS. This renders the selective abort strategy useless, since unopened secrets can always be reconstructed by an honest majority of

parties. On the downside, such protocols require more communication and computation. Although ALBATROSS [9] allows parties to generate a large batch of outputs with little overhead, these outputs are all revealed at once, instead of gradually providing fresh random values, as in TVRF-based protocols.

There are a number of recent works on constructing randomness beacons from time based primitives, such as Time Lock Puzzles (TLP) [6, 19, 25, 31] and Verifiable Delay Functions (VDF) [5, 17, 29, 38]. Such protocols [2, 3, 5] require a structured common reference string as setup and achieve communication complexity linear in the number of parties but are based on sequential computation assumptions, *e.g.* [31] and [17]. These assumptions are arguably less understood than classical assumptions such as the Decisional Diffie-Hellman (DDH) assumption, upon which PVSS and (T)VRF-based beacons can be constructed. Hence, we focus on the latter, basing all our constructions on DDH. However, time-based primitives can potentially be used to instantiate Tier 2 of our beacon.

1.1 Our Contributions

In this work, we aim to combine the PVSS and (threshold) VRF approaches to obtain a best-of-both-worlds “multi-tiered” randomness beacon construction. Moreover, as a key part of Mt. Random’s construction, we design a novel protocol for publicly verifiable and unbiased distributed key generation. Finally we also present GULL (Gradually UnLeashed aLbatross), a new PVSS-based beacon that generates a large batch of random outputs like ALBATROSS but allows for gradually releasing of smaller “sub-batches” of outputs. All of our constructions are publicly verifiable and proven secure against malicious adversaries under a single standard assumption, *i.e.* Decisional Diffie Hellman (DDH).

Mt. Random: A Multi-tiered Randomness Beacon. More precisely, Mt. Random is a protocol where VRF, TVRF and PVSS-based random beacons are run as independent tiers executed in parallel. Each tier offers a different trade-off between complexity and randomness quality. By using the outputs of each tier as seeds for the next one, we aim at constructing a flexible architecture for randomness beacons that achieves good concrete efficiency without sacrificing security guarantees. Moreover, our approach allows for higher level protocols to choose what tier to use when obtaining randomness, according to the best complexity vs. randomness quality trade-off for each application. At a glance, Mt. Random is constructed as follows:

- **Setup - Distributed Key Generation (DKG):** All Tiers use a Public Ledger for communication in order to achieve public verifiability and Tiers 1 and 2 use our novel DKG protocol to obtain threshold key pairs.
- **Tier 1 - Uniform Randomness via PVSS with GULL:** This tier gives $O(n)$ individual batches of $O(n)$ uniformly random outputs with communication and computational complexities of $O(n^2)$ for n parties.
- **Tier 2 - Uniform Pseudorandomness via TVRFs:** This tier uses a fresh output from Tier 1 as a seed when a new one is needed, providing 1 uniformly

pseudorandom output per execution with communication and computational complexities linear in the number of parties running the tier.

- **Tier 3 - Bounded-Biased Pseudorandomness via VRFs:** This tier uses a fresh output from Tier 2 as a nonce when a new one is needed. Communication and computational complexities depend on output bias, *i.e.* the lower the complexity the higher the adversarial bias on the output.

Publicly Verifiable Distributed Key Generation. We introduce a new publicly verifiable distributed key generation (DKG) protocol that can provide both the keys needed for the threshold encryption used in GULL (Tier 1) and for the TVRF (Tier 2). The security of our DKG scheme is based solely on DDH (in the random oracle model) and it guarantees that the public key cannot be biased by a rushing adversary (unlike in some other alternatives). In terms of communication and computation, our protocol is more efficient than previous unbiased DKG schemes and essentially as efficient as the best biasable schemes.

GULL (Gradually UnLeashed aLbatross). We introduce GULL, a PVSS-based random beacon that generates $O(n)$ individual sub-batches of $O(n)$ random outputs each, where n is the number of parties in the protocol. Differently from the previous work ALBATROSS, all sub-batches remain initially hidden and can be opened at different points of time. Opening one sub-batch gives no information about the yet unopened ones. While GULL is marginally slower than ALBATROSS if a full batch of random outputs is required, it is significantly more efficient when a sequence of fresh unpredictable outputs are required: GULL allows for preprocessing a large amount of sub-batches of uniformly random outputs that can be gradually revealed at a low cost.

1.2 Technical Overview

Besides the Mt. Random architecture described above, our main technical contributions are novel protocols for PVSS-based randomness generation (GULL, used for implementing Tier 1) and for distributed key generation (used by both Tiers 1 and 2). We describe our main novel techniques.

Distributed Key Generation. Departing from SCRAPE and ALBATROSS, we construct a DKG secure under the DDH-assumption and compatible with threshold El Gamal and TVRFs. Our goal is to establish a common public key $\text{tpk} = g^{\text{tsk}}$ and partial public keys of the form $\text{tpk}_i = g^{\text{tsk}_i}$, where each party P_i receives secret keys tsk_i that are Shamir shares for tsk . The aforementioned PVSS-based beacons output a group element g^r , that can be set as tpk . This is essentially constructed by having each party deal a secret g^s by constructing Shamir shares σ_i of s and encrypting g^{σ_i} under P_i 's public key. Then g^r is defined by aggregating the g^s that were shared correctly (*i.e.* $g^r = \prod g^{s^{(a)}}$ where $s^{(a)}$ is correctly shared by P_a). In the process of reconstructing g^r , parties obtain partial

public keys tpk_i by decrypting the shares they have received and aggregating them (i.e. $\text{tpk}_i = \prod g^{\sigma_i^{(a)}}$), while proving the validity of the process.

However, parties do not obtain the corresponding partial secret keys sk_i from the PVSSs, as parties only obtain values g^{σ_i} but not σ_i . In our DKG, we modify the secret sharing phase described above, by having dealers also send a ciphertext containing the Shamir share σ_i which can only be decrypted by learning the corresponding “group share” g^{σ_i} . This guarantees that only party P_i can reconstruct σ_i . However, we then need to deal with the case where P_i detects that the group share and Shamir share are inconsistent with each other. In comparison to Fouque-Stern DKG, where the use of Paillier encryption allows the dealer to construct an elegant, but expensive, proof of the fact that the two values are the same, here this is not possible. Instead, we resort to a dispute resolution where the complaining party P_i reveals the received Shamir share and either P_i or the dealer of that share is disqualified. Revealing this share does not harm privacy since one of the two parties will be disqualified and that share will never be used.

One technical novelty we introduce with respect to ALBATROSS, which we will also exploit later in GULL, lies in the order of operations: in ALBATROSS parties first decrypt and reveal each g^{σ_i} , jointly reconstruct the secrets g^s of each dealer, and these opened secrets are aggregated to create the final output g^r ; here, instead, parties aggregate their encrypted shares first and only reveal (and prove correctness of) the aggregated shares, which become the partial keys tpk_i ; then tpk is reconstructed from the tpk_i . This change of order can be done because the operations involved are linearly homomorphic.

GULL. While the ALBATROSS construction provides a large uniformly random output, one problem is that the whole output is reconstructed by the participants at once. In Mt. Random and other applications, it is instead desirable that parts of this output are released gradually, while the rest of the output is kept hidden. We depart from ALBATROSS to construct GULL, a random beacon that accomplishes this. In ALBATROSS, the output consists of $\ell \cdot \ell'$ group elements, that we can group as ℓ' blocks of ℓ elements each; all these blocks are released at once. In GULL, parties execute the beginning of the protocol as in ALBATROSS (until the whole output is fixed), but then can release every block of ℓ outputs independently. Every block release needs little communication and computation and blocks not yet released are unpredictable given the opened ones.

In order to do this, after parties publish their encrypted shares and correctness proofs, and the set \mathcal{Q} of dealers who have shared their secret correctly has been set, every party aggregates their shares received from the different dealers as a first step, then decrypts their aggregated shares, but rather than communicating them, they re-encrypt them under threshold El Gamal encryption, proving this encryption is correct and consistent with the rest of the protocol. We introduce and analyse a protocol π_{EG} for this proof. At this point, there are ℓ' blocks of ℓ secrets shared, where for each block the vector of ℓ secrets is shared via packed Shamir secret sharing, and at least $t + \ell$ receivers have

encrypted their share with threshold El Gamal. Due to linearity of both the El Gamal scheme and the secret sharing reconstruction, the encryptions of the secrets in a block can be computed from any set of $t + \ell$ encrypted shares. Now a large enough subset of parties can decrypt each coordinate of the secret. Unfortunately, decrypting one of these coordinates may give information about the other secret coordinates *in the same block*, so we release all of the block of ℓ coordinates at once. However, the remaining unopened blocks of ℓ coordinates are still secret and uniformly distributed in the view of any subset of t parties.

1.3 Related Works

Table 1. Comparison of DKG schemes where n is the total number of parties, t is the number of corrupted parties, k_q is the number of bits of an element of \mathbb{G}_q or \mathbb{Z}_q , k_N is the number of bits of the Paillier cryptosystem modulus N and k_h is the output length of a hash function. Exp, Enc, Dec stand for operation of \mathbb{G} (*i.e.* exponentiation), Paillier encryption and Paillier decryption, respectively. We consider that Pedersen and Gennaro *et al.* have private messages encrypted under El Gamal. For typical parameters $k_q = 256, k_N = 2048$, we have $k_N = 8k_q$, Enc=3600 Exp and Dec=4880 Exp.

Scheme	Comp. (Exp/Enc/Dec)	Comm. (bits)	Rounds	Bias Resist.	Assump.
Pedersen [28]	$nt + 5n + t + 1$	$(2n^2 + tn + n)k_q$	1 + 2	No	DDH
Gennaro et al. [21]	$2nt + 11n + 3t + 3$	$(4n^2 + 2tn + 2n)k_q$	2 + 3	Yes	DDH
Fouque-Stern [18]	$(nt + 5n + t + 1)$ Exp. +4n Enc+n Dec	$(2n^2 + tn + n)k_q$ $+ 2n^2k_h + 3n^2k_N$	1	No	DDH+DCR
F-S [18] in terms of Exp. and k_q	$nt + 18005n + t + 1$	$(28n^2 + tn + n)k_q$	1	No	DDH +DCR
Our Result	$9n + t + 2$	$(2n^2 + tn + 5n)k_q$	2 + 2	Yes	DDH

Distributed Key Generation. While most distributed key generation (DKG) protocols employ secret sharing similarly to the one we introduce, the key differences lie on how parties prove the correctness of their shares and their consistency with the public information they post. Possibly the best known is Pedersen’s protocol [28], where parties use a Feldman’s VSS to do this, resulting in a protocol with at least 1 round of interaction, and 2 additional rounds if there are disputes. Gennaro et al. [21] observed that malicious parties can bias the public key generated by Pedersen’s DKG and fixed this problem by introducing a new round of interaction and a new round of dispute resolution. Fouque and Stern [18] proposed a publicly verifiable one-round DKG based on the Paillier cryptosystem that still allows the adversary to bias public keys. A recent work by

Gurkan et al. [22] introduces a publicly verifiable DKG with communication complexity of $O(n)$ based on the notion of aggregation via gossip. However, this protocol is based on pairing assumptions, stronger than our DDH assumption, and also outputs group elements as secret keys (rather than elements in \mathbb{Z}_q), *i.e.* the output is *incompatible with threshold El Gamal encryption*. It would be very interesting to achieve the type of output keys we need with gossip techniques.

In Table 1, we compare DKG protocols in terms of computation, communication, number of rounds (fixed rounds+dispute resolution rounds), assumptions and biasability of the global public key. Since Pedersen’s and Gennaro et al.’s protocols need private communication between parties, we assume that this is done through the public ledger using El Gamal encryption. For comparing to Fouque-Stern, we estimated that Paillier encryption and decryption are equivalent to 3600 and 4880 group operations over a DDH-hard group, respectively, at the 128-bit security level, on a Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz using the RELIC library [1]. Our protocol requires almost the same communication as Pedersen’s, differing only in lower order terms, and less communication than Gennaro et al. and Fouque-Stern, especially when compared with the latter, since k_N is larger than k_q (we can currently assume $k_q = 256$, $k_N = 2048$). On the other hand, Pedersen and of course Fouque-Stern have better round complexity, at the cost of allowing bias on the public key.

Table 2. Comparison of random beacon protocols according to computation in terms of modular exponentiations, communication in terms of bits posted on public ledger, bias resistance, computational assumption and setup assumption. n = number of parties, PKI = Public Key Infrastructure, RO = Random Oracle, SCRS = Structured Common Reference String, URS = Uniform Random String, VDF = computational cost of VDF evaluation. In GULL (generation), “Output size” means that a batch of $O(n^2)$ encrypted random elements are prepared but not opened until the Opening phase.

	Computation	Comm. Ledger	Output Size	Bias Resist.	Comp. Assumption	Setup Assumption
GRandPiper [4]	$O(n)$	$O(n^2)$	1	Yes	q-SDH	SCRS
HydRand [33]	$O(n)$	$O(n^2)$	1	Yes	DDH	PKI, RO
ALBATROSS [9]	$O(n^2 \log n)$	$O(n^2)$	$O(n^2)$	Yes	DDH	PKI, RO
GULL (generation)	$O(n^2)$	$O(n^2)$	$O(n^2)$	Yes	DDH	PKI, RO
GULL (opening)	$O(n^2)$	$O(n^2)$	$O(n)$	Yes	DDH	PKI, RO
DRAND [36]	$O(n)$	$O(n)$	1	Yes	Gap-DH	DKG, RO, Many URS
Ouroboros Praos [16]	$O(n)$	$O(n)$	1	No	CDH	PKI, RO, URS
Original VDF [5]	VDF	$O(n)$	1	Yes	VDF	SCRS
RandRunner [32]	VDF	$O(n)$	1	Yes	VDF	URS, RO
Mt. Random-Tier 1	$O(n^2)$	$O(n^2)$	$O(n)$	Yes	DDH	PKI, RO
Mt. Random-Tier 2	$O(n)$	$O(n)$	1	Yes	DDH	PKI, RO, Tier 1
Mt. Random-Tier 3	$O(n)$	$O(n)$	1	No	DDH	PKI, RO, Tier 2

Randomness Beacons. Mt. Random is the first random beacon of its kind, *i.e.* combining different sub-protocols generating fresh random outputs each with a different randomness quality vs. efficiency trade-off. Previous works have focused on improving one specific approach to randomness generation instead of combining multiple approaches. On the other hand, in introducing the novel multi-tiered architecture of Mt. Random the main hurdle was supporting protocols compatible with all Tiers and based on a single assumption (DDH): the new DKG protocol used for Tiers 1 and 2 and the new Tier 1 beacon GULL that allows for efficiently gradually releasing large batches of random outputs. Hence, when comparing with previous works, we focus on comparing each previous work with a specific tier of Mt. Random. Besides communication/computational/round complexity, we also take into consideration the supporting protocols required as setup by previous works and the security assumptions they are based in, since providing a cohesive suite of protocols based on a single assumption is a key feature of Mt. Random. A comparison is presented in Table 2.

In comparing Mt. Random with previous works it is useful to observe the following correspondences between its Tiers and each previous protocol: Tier 1 consists of executing GULL's generation phase and then successively opening individual sub-batches of outputs; Tier 2 consists of running a DDH-based version of DRAND using seeds from Tier 1; Tier 3 consists of running Ouroboros Praos using nonces from Tier 2. Tier 1 has a little overhead with respect to the state-of-the-art beacon ALBATROSS in case $O(n^2)$ outputs are needed at once, but has an advantage within Mt. Random's architecture where sub-batches of fresh $O(n)$ outputs are frequently required. In this case, ALBATROSS would have to be executed n times to obtain the same amount of fresh randomness batches, resulting in complexity much higher than that of Tier 1 using GULL. Tiers 2 and 3 naturally have the same complexity as DRAND and Ouroboros Praos. However, instead of relying on stronger assumptions, an external DKG phase and an external source of fresh seeds as is the case of DRAND, Tier 2 is fully based on DDH and receives keys and fresh seeds from our DKG protocol and Tier 1, which are also based on DDH. Hence, the main advantage of Mt. Random is providing a self-contained infrastructure for executing its upper Tiers based on a single well studied assumption without sacrificing efficiency, which is made possible by our novel DKG and GULL protocols respectively executed for generating threshold keys and running Tier 1.

2 Preliminaries

For integers $m \leq n$, $[m, n]$ denotes the set $\{m, m + 1, \dots, n\}$. We let $[n] = [1, n]$. Our protocols take place in a cyclic group \mathbb{G} of prime order q . We denote by \mathbb{Z}_q the finite field of q elements (for the same prime q), consisting of the integers modulo q , and note that we can speak of g^a for $g \in \mathbb{G}, a \in \mathbb{Z}_q$ and this respects the rule $g^a \cdot g^b = g^{a+b}$ where the sum is in \mathbb{Z}_q . Finally $\mathbb{Z}_q[X]_{\leq d}$ denotes the set of polynomials in $\mathbb{Z}_q[X]$ with degree at most d .

2.1 Adversarial and Communication Models

We consider security against a malicious static adversary, which may arbitrarily deviate from the protocol but chooses what parties to corrupt before the execution starts¹. The adversary will corrupt at most $(n - \ell)/2$ parties, for some integer $\ell > 0$, which we think of as a small fraction of n . For simplicity, we assume access to an authenticated bulletin board. Once a party posts a message to the bulletin board, it becomes immutable and immediately available to all other parties, who can also verify the authenticity of the message (*i.e.* that it was indeed posted by a given party). Such a bulletin board could be substituted by a blockchain based public ledger, a public key infrastructure and digital signatures; however, modeling the corner cases that arise in this scenario introduces a number of technicalities that are not the main focus of this work. We assume synchronous communication: messages sent (or posted to the bulletin board) in a round are guaranteed to be received by all honest parties by the next round. Our protocols can be extended to the partially synchronous setting (*i.e.* with adversarially controlled but finite communication delays) via standard techniques (*e.g.* waiting for a majority of valid shares to be received [2, 3, 9]).

2.2 Packed Shamir Secret Sharing

Secret sharing allows to distribute a secret among n parties P_1, \dots, P_n by delivering a share to each party, so that only certain subsets of these parties can later reconstruct it by pooling together their received shares.

In (t, ℓ) -packed Shamir secret sharing over \mathbb{Z}_q (with q prime), the secret is a vector $(s_0, \dots, s_{\ell-1}) \in \mathbb{Z}_q^\ell$, and order to share this secret, the dealer chooses $f \in \mathbb{Z}_q[X]_{\leq t+\ell-1}$ with $f(-j) = s_j$ for $j \in [0, \ell - 1]$, and sends the evaluation $\sigma_i := f(i)$ as share to party P_i . Here we are assuming that $q \geq n + \ell$, so that the evaluation points $-j$ for the secrets and i for the shares are disjoint. The more classical Shamir secret sharing scheme is the case $\ell = 1$. The (t, ℓ) -packed Shamir secret sharing satisfies t -privacy, meaning that the secret vector is distributed independently from any set of t shares, and $t + \ell$ -reconstruction, *i.e.*, the secret vector can be recovered from any set A of $t + \ell$ shares. For the latter, Lagrange interpolation can be used: each secret coordinate can be reconstructed as a linear combination of the shares in the set. Namely, $s_j = \sum_{i \in A} L_{i,A}(-j) \cdot \sigma_i$ for the Lagrange polynomials $L_{i,A}(X) = \prod_{k \in A, k \neq i} \frac{X-k}{i-k}$.

2.3 Non-interactive Zero Knowledge Proofs

In a zero knowledge proof of knowledge a prover wants to convince a verifier that she knows a piece of information (witness) that makes certain statement true without revealing anything about this witness. Non-interactive proofs carry

¹ This is the model assumed by most random beacon constructions with the only exception (to the best of our knowledge) of [16], and proving security against adaptive adversary would require expensive techniques such as non-committing encryption.

out this with a single message from the prover. We consider proofs for public verifiers, meaning anyone can verify the proof. We need non-interactive zero-knowledge proofs of knowledge for two types of statements in a cyclic group of prime order q : discrete logarithm equality (DLEQ) proofs [11] and low-degree exponent interpolation (LDEI) [9]. In fact, DLEQ proofs can be seen as a special case of LDEI proofs. Both can be realized from Sigma-protocol techniques.

In a LDEI proof, the statement is given by a vector of elements of the group, and the prover claims that their discrete logarithms with respect to a given vector of public generators are evaluations of a polynomial of degree lower than certain bound. If we set this bound to be 0 (i.e. the only accepted polynomials are constants) then we have a DLEQ proof: we are proving that the discrete logarithm of the given elements with respect to the generators are all equal. A non-interactive LDEI proof of knowledge (in the random oracle model) of the interpolating polynomial was presented in [9] and is given in Fig. 1.

Low-degree exponent interpolation (LDEI) ZKPoK
 $\pi_{LDEI}((g_i)_{i=1}^m, (x_i)_{i=1}^m, d)$.

Setup: Finite group \mathbb{G} of prime order q , fixed pairwise distinct elements $\alpha_1, \dots, \alpha_m$ in \mathbb{Z}_q , a random oracle $H : \mathbb{G}^{3m} \rightarrow \mathbb{Z}_q$.

Public inputs: $g_1, \dots, g_m, x_1, \dots, x_m \in \mathbb{G}$, $d \in \mathbb{Z}$, $d \geq 0$.

Statement: The prover knows $w(X) \in \mathbb{Z}_q[X]_{\leq d}$ with $x_i = g_i^{w(\alpha_i)} \forall i \in [m]$.

Protocol:

- The prover samples $u(X) \leftarrow \mathbb{Z}_q[X]_{\leq d}$ and computes $a_i = g_i^{u(\alpha_i)}$ for all $i \in [m]$, $e = H(g_1, \dots, g_m, x_1, \dots, x_m, a_1, \dots, a_m)$, and $z(X) = u(X) - e \cdot w(X)$. The proof is (e, z) .
- The verifier computes $a_i = g_i^{z(\alpha_i)} x_i^e \forall i \in [m]$ and accepts iff $\deg z \leq d$ and $e = H(g_1, \dots, g_m, x_1, \dots, x_m, a_1, \dots, a_m)$.

Fig. 1. LDEI zero knowledge proof of knowledge π_{LDEI} from [9]. $\pi_{DLEQ}((g_i)_{i=1}^m, (x_i)_{i=1}^m)$ will denote the case $d = 0$.

2.4 Publicly Verifiable Secret Sharing (PVSS)

A publicly verifiable secret sharing scheme allows any external party to verify the correct sharing and reconstruction of a secret, with the help of zero knowledge proofs posted respectively by the dealer and the reconstructing parties. We will base our constructions upon techniques from SCRAPE [8] and the subsequent modifications in ALBATROSS [9]. These schemes in turn follow the blueprint of Schoenmakers’ PVSS [34]. The PVSS in ALBATROSS can be seen as a generalization of SCRAPE that allows for a flexible trade-off where the dealer can share a vector of ℓ group elements, while at most $t \leq (n - \ell)/2$ parties can be corrupted if we want both t -privacy and $(n - t)$ -reconstruction, which will be necessary later. In contrast, the parameters in SCRAPE (and in Schoenmakers’

PVSS) would correspond to the case $\ell = 1$. The amortized computation and communication per secret shared in ALBATROSS becomes much better as ℓ grows.

If a party correctly PVSSs a secret, meaning that the (publicly verifiable) proof of correct sharing is valid, then this party is committed to the secret in a way that the commitment can be opened by a large enough set of honest share-receivers, regardless of whether the dealer wants to open it. This allows to construct a commit-and-open random beacon that does not present the problem that parties may decide on opening their commitments depending on other opened commitments they have seen. More precisely, in a PVSS-based random beacon each party PVSSs a random secret and everyone can compute the set \mathcal{Q} of parties that have correctly shared; all the secrets dealt by parties in \mathcal{Q} are then opened and the final output is obtained by applying a randomness extractor to these opened secrets, which guarantees that the result is uniformly random and independent from the inputs of any set of t parties. The randomness extractor in ALBATROSS is built from a t -resilient matrix, which we define next.

Definition 1. *A matrix $M \in \mathbb{Z}_q^{r \times m}$ is t -resilient if for any $A = \{i_1, \dots, i_t\} \subseteq [m]$ of size t , $M\mathbf{v}$ is independent from the coordinates of \mathbf{v} indexed by A , i.e. for any $(y_1, \dots, y_t) \in \mathbb{Z}_q^t$, the distribution of $M\mathbf{v}$ when conditioned to $v_{i_1} = y_1, \dots, v_{i_t} = y_t$ and $(v_j)_{j \notin A}$ being uniform in \mathbb{Z}_q^{m-t} , is uniform in \mathbb{Z}_q^r .*

A t -resilient matrix with the parameters above needs to satisfy $r \leq m - t$. An optimal choice (i.e. $r = m - t$) is to let M be a transpose of a Vandermonde matrix (we are assuming $q \geq m$). For computation efficiency reasons, [9] chooses M with entries $M_{ij} = \alpha^{i \cdot j}$ for some $\alpha \in \mathbb{Z}_q$ of order larger than $r \cdot m$.

In our case, $m = n - t$ (as we can only guarantee $n - t$ parties have correctly shared their inputs) and thus $\ell' = n - 2t$ is the maximum size of the output of the t -resilient function. In [9], parameters were set such that $\ell' = \ell$. In this paper, we keep ℓ and $\ell' = n - 2t$ as two separate parameters.

2.5 Verifiable Random Functions (VRFs)

A verifiable random function (VRF) [27] is a pseudorandom function that can be evaluated by the owner of a secret key, who at the same time gets a proof or correct evaluation that can be verified with the corresponding public key. A VRF scheme consists of three algorithms (λ is a security parameter):

- $\text{KeyGen}(1^\lambda)$: outputs a public and secret key pair (pk, sk) .
- $\text{Eval}(\text{sk}, x)$ is a deterministic algorithm which outputs a pair (y, π) where y is the output of the function and π is a proof.
- $\text{Verify}(\text{pk}, x, y, \pi)$ is a probabilistic algorithm that outputs 0 or 1 (respectively meaning “reject” or “accept” the proof).

It has been observed in [16] that the standard VRF security definition is not sufficient in the randomness beacon setting. Pseudorandomness only holds if the key pair has been honestly generated (i.e. by KeyGen) but when it is generated

VRF from Ouroboros Praos

Setup: Let G be a cyclic group of prime order q , with generator g . Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^{\ell_{VRF}}$ and $H' : \{0, 1\}^* \rightarrow G$ be random oracles. In addition we implicitly need a random oracle $H^* : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ for the DLEQ proof.

Commands:

- $\text{KeyGen}(1^\lambda)$ chooses a uniformly random $\text{sk} \in \mathbb{Z}_q$, sets $\text{pk} = g^{\text{sk}}$ and outputs (pk, sk)
- $\text{Eval}(\text{sk}, x)$ sets $y = H(x, u)$ where $u = H'(x)^{\text{sk}}$. It moreover defines $\pi = (u, \pi_{DLEQ}((g, H'(x)), (\text{pk}, u)))$, the latter being the proof that $g^k = \text{pk}$ and $H'(x)^k = u$ for a common k , in this case $k = \text{sk}$. It outputs (y, π) .
- $\text{Verify}(\text{pk}, x, y, \pi)$ parses $\pi = (u, \pi')$, checks that π' is a correct DLEQ proof for $(g, H'(x)), (\text{pk}, u)$ and checks $y = H(x, u)$. It accepts if all these checks pass.

Fig. 2. VRF with unpredictability under malicious key generation [16].

maliciously the adversary can bias VRF outputs. In VRF-based beacons (e.g. Fig. 3), the adversary can generate its own key pairs maliciously, and hence we require the VRF to be unpredictable under malicious key generation as defined in [16]. A VRF scheme $(\text{KeyGen}(1^\lambda), \text{Eval}(\text{sk}, x), \text{Verify}(\text{pk}, x, y, \pi))$ with unpredictability under malicious key generation is secure if it holds that:

- (complete provability): for every (pk, sk) generated by KeyGen , and every x , if $(y, \pi) = \text{Eval}(\text{sk}, x)$ then $\text{Verify}(\text{pk}, x, y, \pi) = 1$ with overwhelming probability;
- (unique provability): for every x , any $y_1 \neq y_2$, and any proofs π_1, π_2 , then at least one of $\text{Verify}(\text{pk}, x, y_1, \pi_1)$ or $\text{Verify}(\text{pk}, x, y_2, \pi_2)$ output 0 with overwhelming probability.
- (pseudorandomness): no PPT adversary can distinguish between $\text{Eval}(\text{sk}, x)$ and a uniformly random string, even when having chosen x , after seeing pk .
- (unpredictability under malicious key generation) no PPT adversary who generated (pk, sk) arbitrarily can distinguish between $\text{Eval}(\text{sk}, x)$ and a uniformly random string for an unknown uniformly random x .

VRF-based beacon

Setup: An initial seed σ_0 , and a random oracle $H : \{0, 1\}^{\ell_{VRF}} \rightarrow \{0, 1\}^m$.

Beacon:

1. Each P_i executes $\text{KeyGen}(1^\lambda)$ of the VRF obtaining $(\text{pk}_i, \text{sk}_i)$ and publishes pk_i .
2. At round $r = 1, 2, \dots$: Let $m_r = r \parallel \sigma_{r-1}$.
 - (a) Every party P_i computes and publishes $(\sigma_r^i, \pi^i) = \text{Eval}(\text{sk}_i, m_r)$.
 - (b) Each party defines I to be the set of i for which $\text{Verify}(\text{pk}_i, m_r, \sigma_r^i, \pi^i)$ passes and computes $\sigma_r = \bigoplus_{i \in I} \sigma_r^i$. The output of this round is $w_r = H(\sigma_r)$.

Fig. 3. VRF-based beacon from [16].

We describe in Fig. 2 the VRF with unpredictability under malicious key generation from [16]. A randomness beacon based on this VRF was presented in [16], and is described in Fig. 3. The beacon uses an initial seed σ_0 which may come from a CRS or, as will happen in our multi-tiered beacon, as an output from some protocol. The beacon proceeds iteratively as follows: At each round r each party has a key-pair for a VRF and evaluates the VRF on the seed σ_{r-1} obtaining σ_r^i . The parties compute σ_r as the XOR of the correctly computed σ_r^i (which they can check using the verification procedure and the public keys). The output of that round is the hash $H(\sigma_r)$, while σ_r is used as seed for the next round. Note that malicious parties may bias the result by waiting until honest parties publish their evaluations of the VRF and then deciding whether they publish theirs.

2.6 Threshold Verifiable Random Functions (TVRFs)

Analogously to the case of signatures, one can define a distributed notion of verifiable random functions, where each party can compute a partial evaluation, and any $t+1$ valid partial evaluations can be combined to obtain the global evaluation of the VRF. Following [20] we define a TVRF as the tuple of algorithms below, where as usual t denotes the corruption threshold:

- $\text{DistKeyGen}(1^\lambda)$: outputs secret keys $\text{tsk}_i, i \in [n]$, corresponding public partial keys tpk_i and global public key tpk .
- $\text{PartialEval}(x, \text{tsk}_i, \text{tpk}_i)$ is a deterministic algorithm which outputs a pair $m_i = (y_i, \pi_i)$ where y_i is a partial evaluation and π_i is a proof.
- $\text{Combine}(\text{tpk}, \{\text{tpk}_i\}, x, A, (m_i)_{i \in A})$, where $A \subseteq [n]$ with $|A| \geq t+1$, outputs either (y, π) consisting of global evaluation y and a global proof π , or \perp .
- $\text{Verify}(\text{tpk}, x, y, \pi)$ is a probabilistic algorithm that outputs 0 or 1 (respectively meaning “reject” or “accept” the proof).

A TVRF has the following properties:

- Consistency: Given any x , when we apply Combine to any $\geq t+1$ correct partial evaluations $(m_i)_{i \in A}$, we obtain the same y .
- Robustness: If Combine outputs a pair (y, π) , then $\text{Verify}(\text{tpk}, x, y, \pi) = 1$
- Uniqueness: for every x , for any $y_1 \neq y_2$, and any proofs π_1, π_2 , then at least one of $\text{Verify}(\text{tpk}, x, y_1, \pi_1)$ or $\text{Verify}(\text{tpk}, x, y_2, \pi_2)$ output 0 with overwhelming probability.
- Pseudorandomness: roughly, the adversary corrupting t parties cannot distinguish the output of the function from a uniformly random value, even when choosing the input.

We describe in Fig. 4 a DDH-based threshold VRF inspired by a threshold Boneh-Lynn-Shacham (BLS) signatures from in [20]. Notice that the original DRAND/Dfinity TVRF uses actual pairing based threshold BLS signatures in order to achieve compact proofs. Both this construction and the improved GLOW TVRF construction are proven secure in [20] and could serve as a building block for the DRAND/Dfinity beacon. However, we present the DDH based

version for the sake of simplicity and for making it clear that all Mt. Random building blocks can be instantiated from DDH in the ROM. Note that we do not make the instantiation of `DistKeyGen` explicit, as we both introduced our own scheme in Sect. 3 and discussed a number of alternatives in the Introduction.

DDH-based threshold VRF (DDH-DVRF in [20])

Setup: Let G be a cyclic group of prime order q , with generator g . Let $H : \{0, 1\}^* \rightarrow G$ a random oracle. In addition we implicitly need a random oracle $H^* : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ for the DLEQ proof.

Commands:

- `DistKeyGen`(1^λ) The distributed key generation creates $\text{tsk}_i \in \mathbb{Z}_q$ such that $(\text{tsk}_i)_{i=1}^n$ is a valid Shamir sharing of some secret $\text{tsk} \in \mathbb{Z}_q$. It outputs public $\text{tpk}_i = g^{\text{tsk}_i}$ and $\text{tpk} = g^{\text{tsk}}$, and privately tsk_i only to party P_i , for $i \in [n]$.
- `PartialEval`($x, \text{tsk}_i, \text{tpk}_i$): y_i is computed by P_i as $y_i = H(x)^{\text{tsk}_i}$. In addition compute $\pi_i = \pi_{DLEQ}((g, H(x)), (\text{tpk}_i, y_i))$.
- `Combine`($\text{pk}, \{\text{tpk}_i\}, x, A, (y_i, \pi_i)_{i \in A}$): A subset $A' \subseteq A$ is selected such that A' has cardinality $t + 1$ and π_i is accepted for $i \in A'$. Then $y = \prod_{i \in A'} y_i^{L_{i, A'}(0)}$ and $\pi = (y_i, \pi_i)_{i \in A'}$.
- `Verify`(tpk, x, y, π): Parse $\pi = (y_i, \pi_i)_{i \in A'}$, verify all π_i for $i \in A'$, and check whether $y = \prod_{i \in A'} y_i^{L_{i, A'}(0)}$. Output 1 if all checks pass, otherwise output 0.

Fig. 4. DDH-based threshold VRF (DDH-DVRF in [20]).

The DRAND/Definity beacon we alluded to above is given in Fig. 5. Notice that, in the threshold scenario, the pseudorandomness property of the standard definition is sufficient to guarantee that VRF outputs are unbiased because distributed key generation guarantees that keys are correctly generated.

The DRAND/Dfinity beacon

We assume $t \leq (n - 1)/2$, so there are at least $t + 1$ honest parties. We fix an initial seed σ_0 and $H' : \mathbb{G} \rightarrow \{0, 1\}^*$ a hash function.

1. Parties invoke `DistKeyGen` from the TVRF to obtain the keys $(\text{tsk}, \text{tsk}_i, \text{tpk}_i)$.
2. At round $r = 1, 2, \dots$: Let $m_r = r || \sigma_{r-1}$.
 - (a) P_i computes and broadcasts $(y_i, \pi_i) = \text{PartialEval}(m_r, \text{tsk}_i, \text{tpk}_i)$.
 - (b) Each party applies locally `Combine`($\text{pk}, \{\text{tpk}_i\}_{i \in [n]}, m_r, [n], ((y_i, \pi_i))_{i \in [n]}$) obtaining values (y, π) .
 - (c) We define $\sigma_r = y$ (for use in the next round). The output of round r is $z = H'(\sigma_r)$.

Note that at each step, a public verifier can attest the correctness of the computation by running `Verify`(tpk, x, y, π).

Fig. 5. The DRAND/Dfinity beacon.

2.7 Threshold Encryption

A threshold encryption scheme allows to encrypt a message towards a group of receivers, such that the message can be decrypted by any $t + 1$ of them, but not less. A threshold encryption scheme is composed by the following algorithms:

- $\text{DistKeyGen}(1^\lambda)$: outputs secret keys $\text{tsk}_i, i \in [n]$, corresponding public partial keys tpk_i and a global public key tpk .
- $\text{Enc}(\text{tpk}, m)$ outputs a ciphertext E .
- $\text{LocalDec}(\text{tpk}_i, \text{tsk}_i, E)$ outputs a partial decrypted message x_i .
- $\text{GlobalDec}(\text{tpk}, I, \{\text{tpk}_i\}_{i \in I}, \{x_i\}_{i \in I}, E)$, where $I \subseteq [n]$ with $|I| \geq t + 1$, outputs a decrypted message m' or an error \perp .

We describe informally below the properties we want from a threshold encryption scheme, following the work of [14], which we refer to for formal definitions. These are attained by the threshold version of El Gamal encryption that we show in Fig. 6.

- **Completeness**: If the keys have been honestly generated with DistKeyGen , a message m honestly encrypted, and a set I of at least $t + 1$ honest parties have computed correct partial decryptions x_i of the corresponding ciphertexts with their keys, then GlobalDec , taking that ciphertext and the public keys and partial decryptions of I , will output m .
- **Robustness**: Given as inputs 2 subsets I and J of at least $t + 1$ parties and their corresponding partial decryptions of a same ciphertext, if GlobalDec does not reject then it outputs the same message on both inputs with overwhelming probability.
- **IND-CPA against static corruption**: We assume the adversary corrupts a set A of at most t parties at the beginning of the protocol. The scheme is IND-CPA secure if the adversary cannot guess (with success probability non-negligibly larger than $1/2$) the plaintext corresponding to a given ciphertext, even if this a ciphertext encrypts a message from a set of 2 possible messages that the adversary has chosen, and given of course that the adversary knows all the public keys and the secret keys corresponding to A .

3 Distributed Key Generation via PVSS

We introduce a DKG protocol $\pi_{DDH-DKG}$ in Fig. 7 that is publicly verifiable and guarantees the adversary cannot bias the global public key. Moreover, the $\pi_{DDH-DKG}$ can be easily extended to generate more than one threshold key pair (Remark 2) and to refresh existing secret key shares (Remark 1). In the full version of this paper [10], we formally analyse the security of $\pi_{DDH-DKG}$ in the real/ideal simulation paradigm with sequential composition. This paradigm is commonly used to analyse cryptographic protocol security and provides strong security guarantees, namely that several instances of the protocol can be executed in sequence while preserving their security. More details about this model can be found in [7]. Concretely, we prove Theorem 1, which states that $\pi_{DDH-DKG}$ securely implements the functionality $\mathcal{F}_{DDH-DKG}$ in Fig. 8.

Threshold El Gamal encryption scheme.

Setup: Let G be a cyclic group of prime order q , with generator g .

Commands:

- $\text{DistKeyGen}(1^\lambda)$: The distributed key generation creates $\text{tsk}_i \in \mathbb{Z}_q$ such that $(\text{tsk}_i)_{i=1}^n$ is a valid Shamir sharing of some secret $\text{tsk} \in \mathbb{Z}_q$. It outputs public $\text{tpk}_i = g^{\text{tsk}_i}$ and $\text{tpk} = g^{\text{tsk}}$, and privately tsk_i only to party P_i , for $i \in [n]$.
- $\text{Enc}(\text{tpk}, m)$: To encrypt a message $m \in \mathbb{G}$, sample r uniformly at random in \mathbb{Z}_q , and output $E = (g^r, \text{tpk}^r \cdot m) := (c, d) \in \mathbb{G}^2$
- $\text{LocalDec}(\text{tpk}_i, \text{tsk}_i, E)$ outputs $x_i = (y_i, \pi_i)$ where $y_i = c^{\text{tsk}_i}$ and $\pi_i = \pi_{\text{DLEQ}}((g, c), (\text{tpk}_i, y_i))$.
- $\text{GlobalDec}(\text{tpk}, I, \{\text{tpk}_i\}_{i \in I}, \{x_i\}_{i \in I}, c)$ outputs \perp if no more than t DLEQ proofs $\pi_i, i \in I$ pass. Otherwise, it takes a subset $I' \subseteq I$ of cardinality exactly $t + 1$ such that $\pi_{i \in I'}$ are all correct, and computes

$$m' = d \cdot \left(\prod_{i \in I'} y_i^{-L_{i, I'}(0)} \right)$$

Fig. 6. Threshold El Gamal encryption scheme

Theorem 1. *Under the DDH assumption and assuming an authenticated bulletin board, $\pi_{\text{DDH-DKG}}$ securely realizes $\mathcal{F}_{\text{DDH-DKG}}$ in the random oracle model against a malicious static PPT adversary \mathcal{A} corrupting $t \leq \frac{n-1}{2}$ parties.*

Remark 1 (Refreshing partial keys). The protocol can be modified to one that, given a distributed key ensemble $(\text{tpk}, \{\text{tpk}_i\}, \{\text{tsk}_i\})$ in the form above (not necessarily created by our protocol) outputs fresh random partial secret and public keys $\widetilde{\text{tsk}}_i, \widetilde{\text{tpk}}_i$ corresponding to the same global keys tsk, tpk . This is done by having each party P_a share the value $s^{(a)} = 0$ in step 1) of Fig. 7. It is easy to modify the LDEI proof to additionally prove in zero knowledge that the PVSS is indeed a sharing to 0 (in Fig. 1, the prover just chooses $u(X)$ with the additional condition $u(0) = 0$ and the verifier checks that $z(0) = 0$). Modifying the DKG protocol in this way will output the ensemble $(\text{tpk}', \{\text{tpk}'_i\}, \{\text{tsk}'_i\})$ with $\text{tpk}' = 1_{\mathbb{G}}$. Now parties can define $\widetilde{\text{tpk}}_i = \text{tpk}_i \cdot \text{tpk}'_i$ and each party P_i can privately compute $\widetilde{\text{tsk}}_i = \text{tsk}_i + \text{tsk}'_i$.

Remark 2 (Outputting ℓ' key ensembles). Our DKG protocol would correspond to the case $\ell = \ell' = 1$ in the analogy with ALBATROSS, but of course we can also easily adapt the protocol for $\ell = 1, \ell' \geq 1$, where assuming now $t \leq (n - \ell')/2$, we would obtain as output ℓ' independent instances $(\text{tpk}^{(k)}, \{\text{tpk}_i^{(k)}\}, \{\text{tsk}_i^{(k)}\})$, $k \in [\ell']$. The protocol works in the same way until step 4. In step 5 parties P_i compute $\hat{S}_{i,k} = \prod_{a \in \mathcal{Q}} (\hat{S}_i^{(a)})^{M_{k,a}}$, $\sigma_{i,k} = \sum_{a \in \mathcal{Q}} M_{k,a} \sigma_i^{(a)}$ and $S_{i,k} = \prod_{a \in \mathcal{Q}} (S_i^{(a)})^{M_{k,a}}$ for $k = 1, \dots, \ell'$. Then steps 6, 7, 8 are executed independently for each k , where in step 7 parties verify $\hat{S}_{i,k} = \prod_{a \in \mathcal{Q}} (\hat{S}_i^{(a)})^{M_{k,a}}$. Moreover, the refreshing technique (Remark 1) can be easily extended to deal with refreshing ℓ' ensembles.

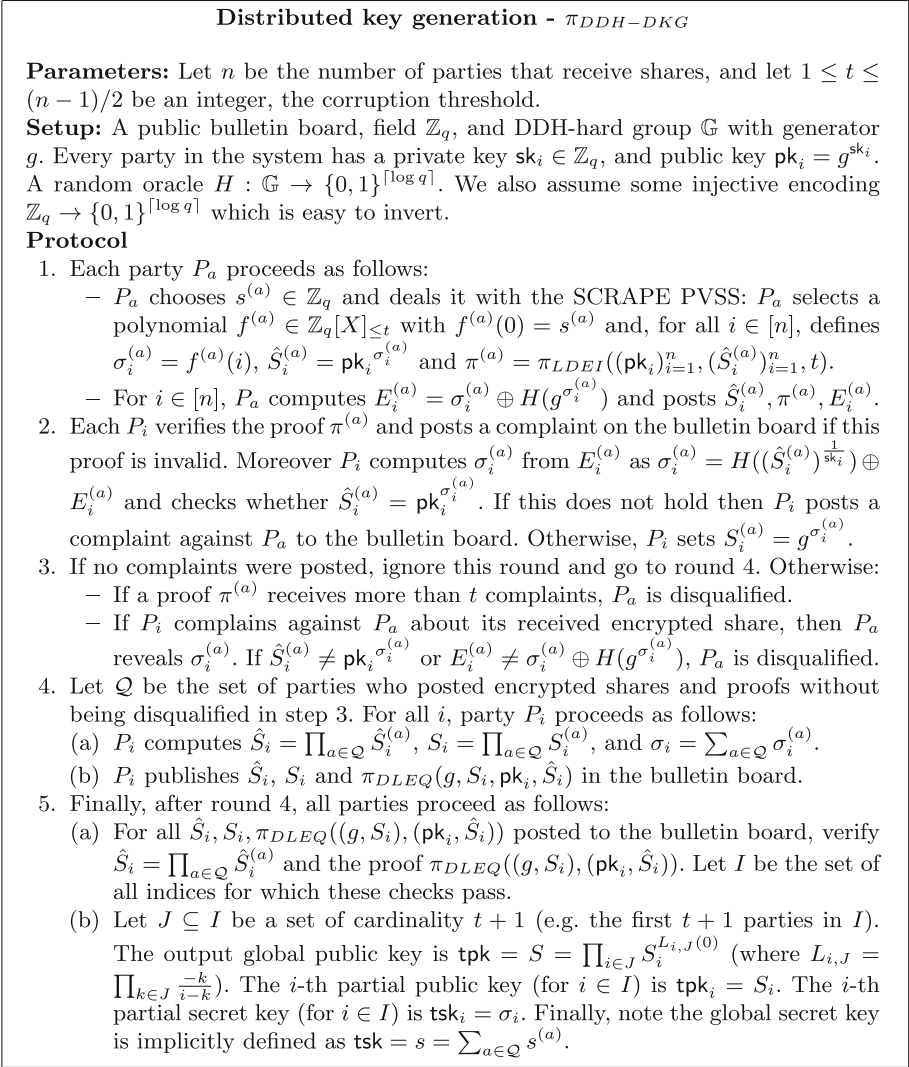


Fig. 7. Protocol $\pi_{DDH-DKG}$ for distributed key generation via SCRAPE.

4 GULL: Gradual Release of PVSS Outputs via Threshold Encryption

Before presenting GULL, we describe a zero-knowledge proof for the EG relation that we will need, which is similar to discrete logarithm equality, except that one of the elements that would be public in the DLEQ relation now is encrypted

Functionality $\mathcal{F}_{DDH-DKG}$

$\mathcal{F}_{DDH-DKG}$ is parameterized by a DDH-hard cyclic group \mathbb{G} of prime order q , with generator g . Let n and $1 \leq t \leq (n - 1)/2$ be integers. $\mathcal{F}_{DDH-DKG}$ interacts with parties P_1, \dots, P_n and an adversary \mathcal{S} that corrupts at most t parties. $\mathcal{F}_{DDH-DKG}$ works as follows:

- Upon receiving $(\text{GEN}, \text{sid}, P_i)$ from a party P_i :
 1. If P_i is honest, forward $(\text{GEN}, \text{sid}, P_i)$ to \mathcal{S} .
 2. If P_i is corrupted, wait for \mathcal{S} to send $(\text{SETSHARE}, \text{sid}, P_i, \sigma_i)$ where $\sigma_i \in \mathbb{Z}_q$ and set $\text{tpk}_i = g^{\sigma_i}$.
 3. Let J be the set of all parties P_j who sent $(\text{GEN}, \text{sid}, P_j)$. If all honest parties are in J , proceed as follows:
 - (a) Sample a random polynomial $f \in \mathbb{Z}_q[X]_{\leq t}$ with $f(i) = \sigma_i$ for all σ_i sent by \mathcal{S} in step 2).^a For every honest party P_h , set $\text{tpk}_h = g^{\sigma_h}$ with $\sigma_h = f(h)$.
 - (b) Set $\text{tpk} = g^{f(0)}$.
 - (c) For corrupted parties $P_c \in J$, send $(\text{KEYS}, \text{sid}, \sigma_c, \{\text{tpk}_j\}_{j \in J}, \text{tpk})$ to \mathcal{S} .
 - (d) Wait for \mathcal{S} to answer with $(\text{ABORT}, \text{sid}, C)$ where C is a set of corrupted parties.
 - (e) For all $j \in J \setminus C$, send $(\text{KEYS}, \text{sid}, \sigma_j, \{\text{tpk}_k\}_{k \in J \setminus C}, \text{tpk})$ to P_j .^b

^a This is possible since the adversary can only set at most t values σ_i .

^b Notice that $\{\text{tpk}_k\}_{k \in J \setminus C}$ can always be used to obtain $\text{tpk} = g^{f(0)}$ by Lagrange interpolation because $|J \setminus C| \geq n - t > t$.

Fig. 8. Distributed Key Generation Functionality $\mathcal{F}_{DDH-DKG}$

by El Gamal (threshold) encryption. The relation is as follows:

$$\mathcal{R}_{EG} = \{((\{\}_{\infty}, \S_{\infty}, \S_{\in}, \sqcup, \rfloor, \lceil), (f, \nabla, \}_{\in})) \in \mathbb{G}^{\times} (\mathbb{Z}_{\Pi}^{\in} \times \mathbb{G}) : g_1^s = x_1, g_1^r = c, d = t^r \cdot g_2, g_2^s = x_2\}$$

The problem here is that g_2 is part of the witness, and should not be revealed. Our solution consists on reducing this to proving knowledge of exponents r, s, w with $g_1^r = c, g_1^s = x_1, d^s \cdot t^w = x_2, c^s \cdot g_1^w = 1$, which can be done with a standard Σ -protocol. The point is that if (s, r, g_2) is a witness for \mathcal{R}_{EG} , then setting $w = -rs$ will satisfy the equations, while on the other hand, knowledge of (r, s, w) satisfying these equations implies knowledge of (r, s, g_2) satisfying \mathcal{R}_{EG} . We present protocol π_{EG} in Fig. 9 and formally state and prove its security in Proposition 1

Proposition 1. *Protocol π_{EG} is a correct proof of knowledge of (s, r, g_2) such that $((g_1, x_1, x_2, t, c, d), (s, r, g_2)) \in \mathcal{R}_{EG}$, with special soundness (with soundness error $1/q$), and zero knowledge in the random oracle model, assuming the Fiat-Shamir heuristic holds.*

Proof. We prove that the interactive public-coin version of this protocol where e is chosen uniformly at random by the verifier is correct, special-sound and zero

NIZK proof of knowledge π_{EG} for \mathcal{R}_{EG}

Setup: Random oracle H . **Common Input:** (g_1, x_1, x_2, t, c, d) . **Witness:** (s, r, g_2)

1. The prover chooses $u_r, u_s, u_w \leftarrow \mathbb{Z}_q$, constructs $a_1 = g_1^{u_r}$, $a_2 = g_1^{u_s}$, $a_3 = d^{u_s} \cdot t^{u_w}$, $a_4 = c^{u_s} \cdot g_1^{u_w}$, creates $e = H(g_1, x_1, x_2, t, c, d, a_1, a_2, a_3, a_4)$ and computes $z_r = u_r + e \cdot r$, $z_s = u_s + e \cdot s$, $z_w = u_w - e \cdot r \cdot s$. The proof is (e, z_r, z_s, z_w) .
2. The verifier computes $a_1 = g_1^{z_r} \cdot c^{-e}$, $a_2 = g_1^{z_s} \cdot x_1^{-e}$, $a_3 = d^{z_s} \cdot t^{z_w} \cdot x_2^{-e}$, $a_4 = c^{z_s} \cdot g_1^{z_w}$ and accepts iff $e = H(g_1, x_1, x_2, t, c, d, a_1, a_2, a_3, a_4)$.

Fig. 9. NIZK π_{EG} for \mathcal{R}_{EG}

knowledge and the Fiat-Shamir heuristic implies the properties above for the non-interactive version.

Correctness: The protocol is easily seen to be correct, as setting $w = -rs$ implies $d^s \cdot t^w = x_2$, $c^s \cdot g_1^w = 1$ if the relation is correct, as argued above, and hence all of the checks will pass.

Special-Soundness: Now suppose that a prover can answer two different challenges $e \neq e'$ with z_r, z_s, z_w and respectively z'_r, z'_s, z'_w . This means that the 4 checks by the verifier pass in both cases. From here it is easy to see that $c^{e-e'} = g_1^{z_r-z'_r}$ and $x_1^{e-e'} = g_1^{z_s-z'_s}$ so one can extract $r = (z_r - z'_r)/(e - e')$, $s = (z_s - z'_s)/(e - e')$ and $g_2 = d \cdot t^{-r}$. Note that these values satisfy that $g_1^s = x_1$, $g_1^r = c$, $d = t^r \cdot g_2$, so in order to show that the extracted (s, r, g_2) is a witness, we only need to additionally show that $g_2^s = x_2$. From the fact that the fourth check passes in both cases, we get that $1 = c^{z_s-z'_s} \cdot g_1^{z_w-z'_w}$, which implies $1 = c^{s(e-e')} g_1^{z_w-z'_w}$. Since we already knew $c = g_1^r$ for the extracted r , this means $g_1^{rs(e-e')+z_w-z'_w} = 1$. Since we are in a group of prime order, so g_1 is a generator, it must hold that $rs(e - e') + z_w - z'_w = 0$. Finally from the fact that the third check passes in both instances we have $x_2^{e-e'} = d^{z_s-z'_s} t^{z_w-z'_w}$, which, using the information deduced in the previous line and the expression for the extracted s , means $x_2^{e-e'} = (d^s t^{-rs})^{e-e'}$. Now since $e - e' \neq 0$ and we are in a group of prime order, this means $x_2 = d^s t^{-rs}$. But the right hand side is exactly g_2^s so $x_2 = g_2^s$ as we wanted to show.

Zero Knowledge: The simulator samples z_r, z_s, z_w, e independently and uniformly at random in \mathbb{Z}_q , and defines a_1, a_2, a_3, a_4 as the verifier would do in the proof verification. This generates a transcript which is indistinguishable from one of an actual protocol, as it is easy to see.

We now construct GULL, a random beacon that allows for generating $O(n^2)$ outputs that can be opened in individual batches of $O(n)$ outputs. We present GULL in Fig. 10 and formally state its security in Theorem 2.

Theorem 2. *Assuming DDH holds in \mathbb{G} , for any static adversary \mathcal{A} corrupting t parties, with probability at least $1 - t/q$ the following holds (in the random oracle model) for protocol GULL in Fig. 10:*

GULL: PVSS beacon with gradual release.

Setup: A public bulletin board, a DDH-hard group $\mathbb{G} = \langle g \rangle$ of prime order q , a t -resilient matrix $M \in \mathbb{Z}_q^{\ell' \times (n-t)}$, key-pairs $(\text{sk}_i, \text{pk}_i = g^{\text{sk}_i}) \in \mathbb{Z}_q \times \mathbb{G}$ for each P_i .

Setup from DKG: Parties have a global threshold public key tpk , partial threshold keys tpk_i and partial threshold secret keys tsk_i , where $\text{tpk}_i = g^{\text{tsk}_i}$ and $\text{tsk}_i = h(i)$ for some secret random $h \in \mathbb{Z}_q[X]_{\leq t+1}$, and $\text{tpk} = g^{h(0)}$.

Preparation:

1. (ALBATROSS Sharing) Each P_a PVSSs a secret $(g^{s_0^{(a)}}, \dots, g^{s_{\ell-1}^{(a)}}) \in_R \mathbb{G}^\ell$ by choosing $f^{(a)} \in_R \mathbb{Z}_q[X]_{\leq t+\ell-1}$ setting $s_j^{(a)} := f^{(a)}(-j)$, $\sigma_i^{(a)} := f^{(a)}(i)$ and publishing $\hat{S}_i^{(a)} = \text{pk}_i^{\sigma_i^{(a)}}$ for $i \in [n]$, together with a proof of correct sharing $\pi^{(a)} = \pi_{LDEI}((\text{pk}_i)_{i=1}^n, (\hat{S}_i^{(a)})_{i=1}^n, t + \ell - 1)$.
2. (a) (Verification) Parties check the validity of $\pi^{(a)}$ for all $a \in [n]$. This fixes a set \mathcal{Q} of the first $n - t$ parties $P_a, a \in \mathcal{Q}$ whose $\pi^{(a)}$ are correct.
 (b) (Aggregation) Every party can compute, for every $i \in [n]$ and $k \in [1, \ell']$,

$$R_{ik} = \prod_{a \in \mathcal{Q}} (\hat{S}_i^{(a)})^{M_{k,a}}$$

Additionally each P_i computes $S_{ik} = R_{ik}^{\text{sk}_i^{-1}}$ for every $k \in [1, \ell']$. Note $S_{ik} = g^{f_k(i)}$ where $f_k(X) = \sum_{a \in \mathcal{Q}} M_{k,a} \cdot f^{(a)}(X)$.

- (c) (Encryption) For every $k \in [\ell']$, P_i posts $E_{ik} = \text{Enc}(\text{tpk}, S_{ik}) = (g^{r_{ik}}, \text{tpk}^{r_{ik}} \cdot S_{ik}) := (c_{ik}, d_{ik})$ and a NIZK proof $\pi_{EG,i,k}$ (detailed in Figure 9) for the fact that $((g, \text{pk}_i, R_{ik}, \text{tpk}, c_{ik}, d_{ik}), (\text{sk}_i, r_{ik}, S_{ik}))$ is in \mathcal{R}_{EG} , i.e. $g^{\text{sk}_i} = \text{pk}_i$, $g^{r_{ik}} = c_{ik}$, $d_{ik} = \text{tpk}^{r_{ik}} \cdot S_{ik}$, $S_{ik}^{\text{sk}_i} = R_{ik}$.

Opening: Let I be the set of the first $t + \ell$ parties P_i who have posted correct proofs $\pi_{EG,i,k}$ for every $k \in [\ell']$. At any point after round 2 is finished, the k' -th batch (for an arbitrary $k' \in [\ell']$) of ℓ outputs can be opened as follows:

- 3) (Lagrange computation) For every $j \in [\ell]$, parties compute: $O_{k',j} = (\prod_{i \in I} (c_{ik'})^{L_{i,I}(-j)}, \prod_{i \in I} (d_{ik'})^{L_{i,I}(-j)})$
- 4) (Decryption) Parties threshold-decrypt $O_{k',j}$ for every $j \in [0, \ell - 1]$ to obtain output $(o_{k',0}, \dots, o_{k',(\ell-1)})$. More concretely, call $O_{k',j} = (C_{k',j}, D_{k',j})$. Then:
 - Each party P_i posts the values $u_{k',j,i} = C_{k',j}^{\text{tsk}_i}$, for all $j \in [0, \ell - 1]$ and a proof $\pi_{k',j,i} = \pi_{DLEQ}((g, C_{k',0}, \dots, C_{k',\ell-1}), (\text{tpk}_i, u_{k',0,i}, \dots, u_{k',\ell-1,i}))$.
 - Let J be a set of the first $t + 1$ parties that have posted correct $\pi_{k',j,i}$. Then the outputs $o_{k',j}, j \in [0, \ell - 1]$ are reconstructed by every party as

$$o_{k',j} = D_{k',j} \cdot \prod_{i \in J} u_{k',j,i}^{-L_{i,J}(0)}$$

Fig. 10. GULL: PVSS beacon with gradual release. Setup and Preparation

- All honest parties obtain the same output $(o_{k,j})_{k \in [\ell'], j \in [0, \ell-1]} \in \mathbb{G}^{\ell \cdot \ell'}$.
- (Unbiasability) Regardless of the actions of \mathcal{A} , the distribution of the output is computationally indistinguishable from uniform in $\mathbb{G}^{\ell \cdot \ell'}$.
- (Unpredictability after opening k' batches –inspired by [9, 24]). For every $k' \in \ell'$, consider the following experiment played after the values $o_{k,j}, k \in [k'], j \in [0, \ell-1]$ have been opened in step 4. A challenger chooses $b \in \{0, 1\}$ at random;

if $b=0$, it sets $w_{k,j} = o_{k,j}$ (the true unopened outputs) for $k \in [k' + 1, \ell']$, $j \in [0, \ell - 1]$; if $b = 1$, it chooses all these $w_{k,j}$ independently and uniformly at random in \mathbb{G} . The challenger sends all $w_{j,k}$ to \mathcal{A} who makes a guess b' . Then the probability that $b' = b$ is at most $1/2 + \text{negl}(\lambda)$ where λ is the security parameter.

Proof. We first note that if a corrupted party cheats in one of the zero knowledge proofs, that party will be caught with probability at least $1 - 1/q$. Therefore the probability that a corrupted party deviates from the protocol and yet is included in \mathcal{Q}, I or J is at most t/q . Hence, except with this probability, all parties included in \mathcal{Q}, I, J have behaved honestly in the respective steps (sharing, threshold encryption, threshold decryption). We assume this from now on. Since there are at least $t + \ell$ honest parties, $|I| \geq t + \ell$ and $|J| \geq t + 1$, so the protocol always finalizes. In fact, the outputs are already determined at the end of step 1: they are $o_{k,j} = g^{f_k(-j)}$, where $f_k = \sum_{a \in \mathcal{Q}} M_{k,a} f^{(a)}$ where \mathcal{Q} is fixed at the end of step 1. Since up to this point the protocol is exactly as in Albatross the unbiasedness under DDH follows from the results there.

We now argue unpredictability. For the sake of notation, assume that the adversary \mathcal{A} corrupts $P_{n-t+1}, \dots, P_{n-t}$. Suppose that after the first k' batches $(o_{k,j})_{k \in [k'], j \in [0, \ell-1]}$ have been opened, \mathcal{A} is given a vector v_b for uniformly random $b \in \{0, 1\}$ where $v_0 = (o_{k,j})_{k \in [k'+1, \ell'], j \in [0, \ell-1]}$ and v_1 is uniformly random in $\mathbb{G}^{\ell \cdot (\ell - k')}$. We will prove that if \mathcal{A} can guess b with probability non-negligibly larger than $1/2$, then we can construct \mathcal{D} that solves the $((n - t) \cdot \ell)$ -DDH problem: namely $(h, h^\alpha, h^{\beta_{11}}, \dots, h^{\beta_{(n-t)\ell}}, h^{\gamma_{11}}, \dots, h^{\gamma_{(n-t)\ell}})$ is sampled as a challenge, where h is uniformly random in the group \mathbb{G} and α , and all β_{ij} are chosen independently and uniformly at random in \mathbb{Z}_q while the value of γ_{ij} is dictated by a bit b' chosen uniformly at random: if $b' = 0$, then $\gamma_{ij} = \alpha\beta_{ij}$ for all i, j ; while if $b' = 1$, γ_{ij} are uniformly random in \mathbb{Z}_q if $b' = 1$. This challenge is sent to \mathcal{D} , who has to guess b' . The $((n - t) \cdot \ell)$ -DDH problem is equivalent to DDH as long as the number $(n - t) \cdot \ell$ is polynomial in the security parameter.

We construct a distinguisher \mathcal{D} which runs an internal copy of \mathcal{A} . \mathcal{D} will simulate an execution of GULL where $g = h^\alpha$ and the vector of secrets chosen by each honest party P_a is $g^{\beta_{aj}}$ (we will show how this can be done). These equal $h^{\gamma_{aj}}$ if $\gamma_{aj} = \alpha\beta_{aj}$. At the end of the simulation, \mathcal{D} sets as a challenge for \mathcal{A} the outputs that one would obtain if the honest parties had shared the values $h^{\gamma_{aj}}$. \mathcal{A} now makes a guess of whether these are correct outputs or random values. In the former case \mathcal{D} guesses that $\gamma_{aj} = \alpha\beta_{aj}$ and in the latter that γ_{aj} are random.

The simulation is as follows: \mathcal{D} sets $g = h^\alpha$, samples u_i in \mathbb{Z}_q for $i \in [n - t]$ and sets $\text{pk}_i = h^{u_i}$, implicitly defining $\text{sk}_i = u_i/\alpha$ (which \mathcal{D} does not know). It waits for \mathcal{A} to choose pk_i and sk_i for malicious parties. Moreover, the distributed key generation algorithm is run to establish the threshold keys $\text{tsk}_i, \text{tpk}, \text{tpk}_i$.

\mathcal{D} chooses τ_{ai} , at random in \mathbb{Z}_q for $a \in [n - t], i \in [n - t + 1, n]$. Let $f^{(a)}$ be the polynomial in $\mathbb{Z}_q[X]_{\leq t + \ell}$ with $f^{(a)}(i) = \tau_{ai}$ for $i \in [n - t + 1, n]$ and $f^{(a)}(j) = \beta_{aj}$ for $j \in [0, \ell - 1]$. \mathcal{D} does not know $f^{(a)}$ but it can compute $h^{f^{(a)}(i)}$, for $i \in [n - t]$, from the values $h^{\beta_{aj}}$ for $j \in [0, \ell - 1]$ and $h^{\tau_{ai}}$ for $i \in [n - t + 1, n]$ by Lagrange interpolation. Note that $h^{\beta_{aj}}$ are part of the challenge. \mathcal{D} sets $\hat{S}_i = (h^{f^{(a)}(i)})^{u_i}$ for

$i \in [n - t]$ and $\hat{S}_i = \text{pk}_i^{\tau_{ai}}$ for $i \in [n - t + 1, n]$. Now $\hat{S}_i = \text{pk}_i^{f^{(a)}(i)}$ for all i , hence it is a sharing for the vector $(g^{f^{(a)}(-j)})_{j \in [0, \ell - 1]} = (g^{\beta_{aj}})_{j \in [0, \ell - 1]}$. \mathcal{D} simulates the proof $\pi^{(a)}$ using the zero knowledge simulator. It waits for the adversary to send the corresponding information from Round 1, which determines \mathcal{Q} . Let $\mathcal{Q}_0 = \mathcal{Q} \cap [1, n - t]$, $\mathcal{Q}_1 = \mathcal{Q} \cap [n - t + 1, n]$.

\mathcal{D} samples values S'_{ik} at random (which will play the role S_{ik} play in the real protocol) and simulates the proof $\pi_{EG,i,k}$. From here, the rest of the protocol is simulated with \mathcal{D} playing for the honest parties and \mathcal{A} for the corrupted ones. After opening k' batches, \mathcal{A} is given as a challenge the values x_{kj} defined by $x_j = \prod_{a \in \mathcal{Q}_0} (h^{\gamma_{aj}})^{M_{k,a}} \cdot \prod_{a \in \mathcal{Q}_1} (g^{f^{(a)}(-j)})^{M_{k,a}}$, where $f^{(a)}$ was extracted from $\pi^{(a)}$. These would be the outputs obtained in the real protocol if $\gamma_{aj} = \alpha\beta_{aj}$, since the secrets of honest parties P_a are implicitly defined as $g^{\beta_{aj}} = h^{\alpha\beta_{aj}}$. While, if γ_{aj} are uniformly random, x_{kj} are also uniformly random due to the properties of the t -resilient matrix.

\mathcal{A} makes a guess about whether the vector of x_{kj} are the secrets or random values, and \mathcal{D} outputs the same guess about whether $\gamma_j = \alpha\beta_j$. Note x_{kj} are the actual secrets if and only if $\gamma_{aj} = \alpha\beta_{aj}$. So if \mathcal{A} 's view $View_{\mathcal{A},k'}$ at this point is as in a real protocol where honest parties have input $g^{\beta_{aj}}$, we would be done.

We argue this by induction. Consider first $k' = 0$, i.e. no outputs have been opened yet. Then apart from $\pi^{(a)}$, $\pi_{EG,i}$, which have been simulated with the zero knowledge simulator (so they are indistinguishable from the real view), the other point where the simulation differs from the real protocol is in the fact that S'_{ik} are chosen at random. But by IND-CPA property of El Gamal (in turn based on DDH) their encryptions are indistinguishable from encryptions of the "real" S_{ik} . For $k' > 0$, the situation is more delicate because now also the opened outputs in the simulated run depend on the simulated S'_{ik} . By induction, the view $View_{\mathcal{A},k'-1}$ of the adversary before the opening of the k' -th output batch is indistinguishable from that in the real protocol. We have argued that, under those conditions and the DDH assumption, our unpredictability claim is true and hence the distribution of the remaining outputs, in particular of the next batch $(o_{k',j})_{j \in [0, \ell - 1]}$, conditioned to that view is indistinguishable from uniform in both the real and simulated run. We conclude that the views $View_{\mathcal{A},k'}$ in the real and simulated protocol are also indistinguishable which finalizes the proof.

5 Constructing Mt. Random

In Fig. 11, we present Mt. Random, our multi-tiered beacon composed by the building blocks presented so far. As discussed earlier, we have three tiers: Tier 1 - Uniform Randomness, Tier 2 - Pseudorandomness and Tier 3 - Bounded Biased Randomness. Starting from Tier 1, going up each tier represents a trade-off between efficiency and randomness quality, where more efficiency is gained at the cost of quality. In other words, higher tiers generate random outputs faster than lower tiers albeit with losses in randomness quality, *i.e.* going from uniformly random values to values with a bounded adversarial bias. Moreover,

Mt. Random: Multi-tiered Randomness Beacon

Parameters:

- n participants $P_i, i \in [n]$, corruption threshold $1 \leq t \leq (n - \ell)/2$.
- Integers $\ell \geq 1$ (number of secrets in GULL output block) and $\ell' = n - 2t$ (number of blocks outputted by one round of GULL).
- Integers ℓ_{TVRF} and ℓ_{VRF} denoting the bitlength of outputs from Tiers 2 and 3.
- Integer $\text{TVRF}_{max} \geq 0$ and $\text{VRF}_{max} \geq 0$ denoting, respectively, the number of times the TVRF-based beacon at Tier 2 and the VRF-based beacon at Tier 3 are applied iteratively starting from a given seed.

Setup: An authenticated public bulletin board (BB), field \mathbb{Z}_q , and DDH-hard group \mathbb{G} with generator g . A key pair $\text{sk}_i \in \mathbb{Z}_q, pk_i = g^{\text{sk}_i}$ registered in BB for each P_i . A t -resilient matrix $M \in \mathbb{Z}_q^{\ell' \times (n-t)}$. Random oracle $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$.

Initialization: All parties P_i keep an initially empty table **Batch** that stores unopened GULL sub-batches encrypted under threshold-El Gamal. All parties first execute the **Distributed Key Generation** phase and then execute **Tier 1**, **Tier 2** and **Tier 3** as soon as seed randomness from the previous tier is available. Tiers are re-executed as more outputs are needed.

Distributed Key Generation: All parties execute the DKG protocol in Figure 7 twice to obtain keys for Tiers 1 and 2. The public outputs are global threshold public keys tpk, tpk' and partial threshold public keys $\text{tpk}_i, \text{tpk}'_i$ for $i \in [n]$, while each party $P_i, i \in [n]$ obtains partial threshold secret keys tsk_i and tsk'_i .

Tier 1: Using keys tpk and tsk_i obtained in the DKG phase, all parties execute the Preparation phase of GULL in Figure 10. At this point all parties obtain ℓ' blocks $B_k = (O_{k1}, O_{k2}, \dots, O_{k\ell}), k \in [\ell']$ consisting of threshold El-Gamal encryptions of o_{kj} under tpk , which are stored in **Batch**. When an output is requested:

1. If **Batch** is not empty, parties execute the Opening phase of GULL to decrypt the next $B_k \in \text{Batch}$, obtaining $o_{k1}, o_{k2}, \dots, o_{k\ell}$. Output $H(o_{k1}), H(o_{k2}), \dots, H(o_{k\ell})$ and remove B_k from **Batch**.
2. If **Batch** is empty, return \perp and run the Preparation phase of GULL to refill it.

Tier 2: Initially, parties request an output o_{kj} from Tier 1 (repeating the request until $o_{kj} \neq \perp$) and set $\sigma_0 = o_{kj}, r = 1$. When an output is requested, all parties execute the DRAND/Dfinity beacon described in section 2.6 using $\text{tpk}', \text{tpk}'_i, \text{tsk}'_i$ with seed $\sigma_0 = o_{kj}$ to obtain output z_r , outputting $H(z_r)$. When $r = \text{TVRF}_{max}$, get a new output o_{kj} from Tier 1 and set $\sigma_0 = o_{kj}, r = 1$.

Tier 3: All parties request an output z_r from Tier 2 (repeating the request until $z_r \neq \perp$) and run the VRF-based beacon in Figure 3 using z_r as initial seed. In each round $r' \in \{1, \dots, \text{VRF}_{max}\}$, output $H(w'_{r'})$ where $w'_{r'} \in \{0, 1\}^{\ell_{VRF}}$ is the output of the beacon. When $r' = \text{VRF}_{max}$, r is reset to 0 and Tier 3 is started again.

Fig. 11. Mt. Random: Multi-tiered Randomness Beacon.

each higher tier uses outputs from the previous tier as seeds, ensuring that all tiers operate within a desired level of bias while maintaining efficiency. We use the DDH assumption (in the random oracle model) to prove security of all of Mt. Random’s building blocks, *i.e.* PVSS, DKG, TVRF and VRF, thus obtaining a final construction whose security is based on a single standard assumption while achieving competitive concrete efficiency. However, we remark that

other constructions of these building blocks can be used within our framework in order to achieve better efficiency at the cost of having security underpinned by multiple and possibly less standard assumptions. Moreover, each Tier could be constructed from other primitives that yield random outputs with similar guarantees, *e.g.* Tier 2 could be instantiated using VDF based beacons [5]. We will now discuss the building blocks used for tier and provide a security analysis.

Tier 1: Uniform Randomness via PVSS: The first tier of Mt. Random outputs true uniform randomness. It is important to output uniformly random values at this tier because these values will be used as seeds for Tier 2. We instantiate Tier 1 with GULL using threshold encryption keys generated by our DKG protocol (Fig. 7). Tier 1 has the highest execution time and communication, outputting uniformly random values less frequently than higher tiers. On the other hand, instead of outputting a single value, Tier 1 will output a *batch* of uniformly random values that can be used to seed Tier 2 multiple times (instead of requiring a full execution of Tier 1 every time Tier 2 needs a new seed).

In the original ALBATROSS [9] protocol, the full batch of outputs is revealed as soon as the protocol terminates. This is not an issue when seeding Tier 2, since Tier 2 outputs cannot be predicted without a threshold key. However, it might be a problem in the case where fresh uniformly random outputs from Tier 1 are required for applications other than seeding Tier 2. Hence, we instantiate Tier 1 with GULL, which allows for gradually revealing smaller “sub-batches” of outputs. Under this regime, whenever a fresh uniformly random output is required for other applications, a fresh sub-batch can be revealed, which is significantly more efficient than re-executing the full ALBATROSS protocol.

Tier 2: Pseudorandomness via Threshold VRFs: Tier 2 outputs pseudorandom values instead of truly uniformly random values. While these values are not suitable for some applications (*e.g.* seeding PRGs), they are sufficient for a number of popular applications (*e.g.* selecting random committees). We instantiate Tier 2 with a DDH based version of the DRAND/Dfinity TVRF proposed in [20] coupled with our new DKG protocol (Fig. 7). We choose to use a DDH-based TVRF in order to instantiate all of our building blocks from a single standard assumption. However, a more efficient TVRF (*e.g.* GLOW [20]) can be used for better performance at the cost of a stronger assumption.

There are two main hurdles in using TVRF-based beacons: 1. keys must be generated in a distributed manner; 2. being essentially a distributed PRG, the beacon must be re-seeded periodically. Mt. Random respectively solves these issues by employing our new DDH-based DKG (Fig. 7) and by periodically re-seeding Tier 2 with uniformly random outputs from Tier 1. Using our DKG, we maintain public verifiability of threshold key validity and consequently of Tier 2’s output without requiring extra assumptions. Moreover our DKG protocol can be used to refresh secret key shares if parties are compromised (see Remark 1).

Tier 3: Bounded Biased Randomness via VRFs: The third tier of Mt. Random outputs pseudorandom values that may be biased by the adversary up to a certain upper bound. While this sort of biased randomness finds less

applications than unbiased pseudorandomness or uniform randomness, it is still sufficient for important applications such as selecting block creators in Proof-of-Stake based blockchains (e.g. Ouroboros Praos [16]). We instantiate Tier 3 with the VRF and VRF-based beacon protocols from Ouroboros Praos, which are secure under the CDH assumption (implied by DDH). However, differently from the original Ouroboros Praos beacon, where each execution is seeded with the output of the previous one, we seed this protocol with an output from Tier 2. This crucial difference reduces the potential adversarial bias in Tier 3 outputs.

5.1 Security Analysis

Theorem 3. *Under the DDH assumption in \mathbb{G} , for any static adversary corrupting $t < n/2$ parties, with probability at least $1 - t/q$, the following holds (in the random oracle model) for Mt. Random in Fig. 11:*

- All Tiers have guaranteed output delivery, all honest parties obtain the same outputs $H(o_{k1}), H(o_{k2}), \dots, H(o_{k\ell}), H(z_r)$ and $H(w'_r)$ from each output request from Tiers 1, 2 and 3, respectively.
- (Unbiasability for Tiers 1 and 2) Regardless of the actions of the adversary, the distribution of $H(o_{k1}), H(o_{k2}), \dots, H(o_{k\ell})$ (resp. $H(z_r)$) from Tier 1 (resp. Tier 2) is computationally indistinguishable from uniform.
- (Unpredictability for Tiers 1 and 2). Given all previous outputs from Tiers 1 and 2, the adversary cannot predict future outputs from Tiers 1 and 2.
- (Bounded Bias and Predictability for Tier 3) The adversary can predict and bias at most t bits of the output of Tier 3.

Proof. (Sketch) Notice that in the initialization phase we execute our DKG protocol (Fig. 7) before initiating the execution of the tiers. Due to the security of the DKG protocol (Theorem 1), the resulting global and partial public keys tpk, tpk_i and $\text{tpk}, \text{tpk}'_i$ for $i \in [n]$ are guaranteed to be unbiased and each party \mathcal{P}_i is guaranteed to have obtained its secret share $\text{tsk}_i, \text{tsk}'_i$ as well as the same public keys. Moreover, we can extract adversarial secret keys, see [10].

In Tier 1, we execute GULL from Fig. 10 using keys $\text{tpk}, \text{tpk}_i, \text{tsk}_i$, which gives us two main guarantees as shown in Theorem 2: 1. The Preparation phase results in ℓ' output blocks that are guaranteed to be recoverable by a majority of the parties but remain secret until the Opening phase is executed; 2. All ℓ values of each output block are guaranteed to be uniformly random. Hence, when Tier 1 is initiated, ℓ' output blocks with ℓ uniformly random values become available. When an output is requested, executing the procedures of Tier 1 clearly returns either a uniformly random output (or \perp , in which case more output blocks are obtained executing step 2 of Tier 1's output request procedure).

In Tier 2, we execute the TVRF-based beacon protocol from Fig. 5, which is proven to output pseudorandom values in [20]. Since we periodically re-seed this protocol with uniformly random values from Tier 1, its outputs are guaranteed to be pseudorandom. Notice that we can re-seed Tier 2 with outputs from Tier 1 that are already revealed but still not used as a Tier 2 seed. By the security

of the TVRF scheme (proven in [20]), an adversary controlling a minority of the parties cannot predict the output of the TVRF on any given input. Hence, the outputs of Tier 2 cannot be predicted by the adversary (who only corrupts a minority of the parties) upon learning the seed. Notice that the TVRF security properties hold since we use unbiased threshold keys tpk' , tpk'_i , tsk'_i .

In Tier 3, we execute the protocol in Fig. 3, proven to output bounded biased values in [16] even when it is seeded with outputs of a previous execution of itself. Hence, seeding this protocol with the unbiased pseudorandom outputs from Tier 2, not only preserves but improves on the proven bias bounds for its outputs. Using outputs from Tier 2 that are already known but still not used as a seed in Tier 3 preserves the security, since even by knowing the seed in advance the adversary can introduce a bounded bias to the output as proven in [16].

6 Efficiency Analysis

Distributed Key Generation. Our novel DKG protocol’s performance is further showcased in Figs. 12a and 12b, which show the DKG computation time and communication size for changing number of parties n for Tiers 1 and 2.

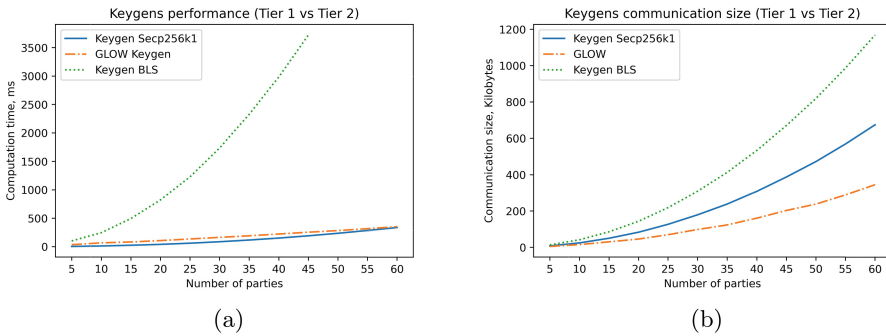


Fig. 12. a) DKG computation time for Tiers 1 and 2 for changing number of parties n with fixed $t = \lfloor \frac{n}{3} \rfloor$. b) DKG communication size for Tiers 1 and 2 for changing number of parties n with fixed $t = \lfloor \frac{n}{3} \rfloor$

Mt. Random. We provide a reference implementation for each one of the tiers here: <https://github.com/ZenGo-X/random-beacon>. Our main goal is to demonstrate the trade-off in efficiency between the three tiers. We also highlight the sensitivity to changing number of parties n , the threshold t and culprits c when relevant. All our measurements were done on a t3.medium AWS instance (2 vCPU of Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50 GHz, 4 GB RAM). Our experiments do not include network latency or delay, as network latency is larger than our computation times and would mask them. Since the number of

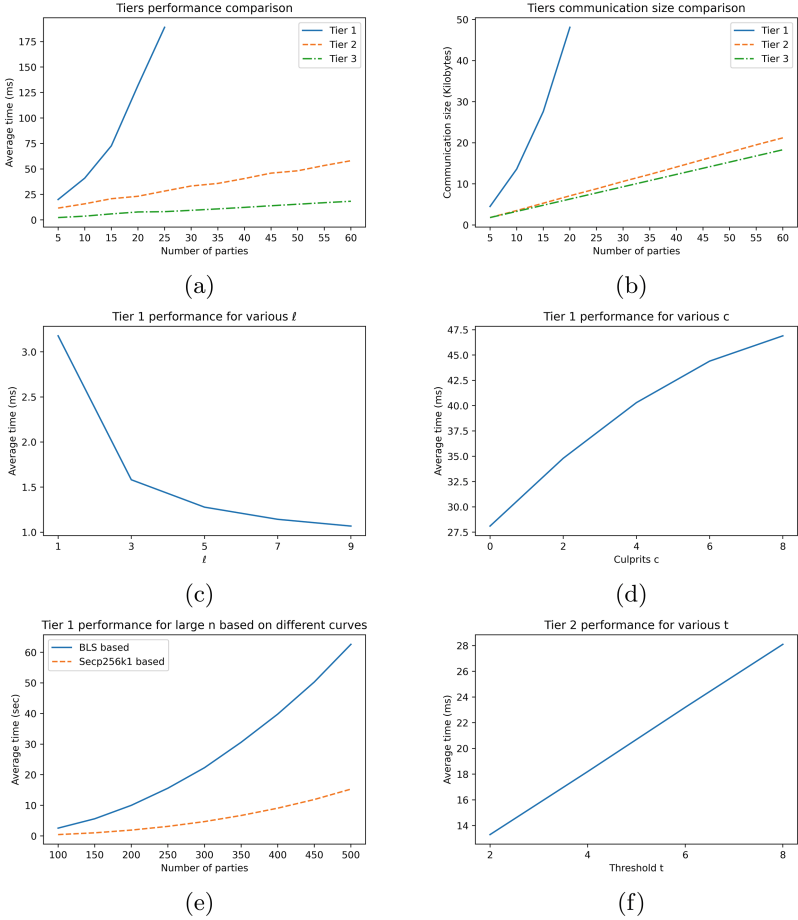


Fig. 13. Execution time (a) and communication (b) of each Tier with fixed $t = \lfloor \frac{n}{3} \rfloor$, $\ell = 1$. (c) Amortized cost of generating a single element at Tier 1 with fixed $n = 25$, $t = 8$, $\ell = 9$. Average execution time of Tier 1 for a number c of corruptions with fixed $n = 25$, $t = 8$ (d) and for large n with fixed $t = \lfloor \frac{n}{3} \rfloor$, $\ell = 1$ (e). (f) Average execution time of Tier 2 for threshold t with $n = 25$.

rounds of Tier 1 is larger than that in Tier 2 and Tier 3, and communication size of Tier 2 is larger than communication size of Tier 3, if we include latency, we trivially get our expected hierarchy. Network delay is of no interest: the communication bandwidth is small enough for the network to not be a bottleneck. Measurements were done using a benchmark tool and averaged over many runs.

Computation Time and Communication Size: In Fig. 13a and Fig. 13b we compare the computation time for a single run and the communication size of each tier as a function of n . As shown by the figures, Tier 1 is the slowest and the one that requires the most communication, while Tier 3 requires the least

computation time and communication, and Tier 2 is in the middle, which is coherent with our use of the tiers in Mt. Random.

Tier 1 and Tier 2 Sensitivity: We measured Tier 1 without gradual release (ALBATROSS), i.e., all random values are released at once. In Fig. 13c we show how changing ℓ impacts the amortized cost of a single random element. As expected, the more random elements we pack in a single run the more efficient the amortized computation per a single random element is. This result hints to the effectiveness of running GULL in settings where fresh unpredictable output is needed by an application other than Tier 2.

In Fig. 13d, we fix n and change the number of culprits c , which impacts the total running time in a meaningful way, since less computation is done in an optimistic case with less misbehaving parties. Figure 13e shows Tier 1 performance for large n . The choice of the curve dramatically affects efficiency. In this case, the Secp256k1 curve implementation outperforms BLS12-381 (we used libraries <https://github.com/rust-bitcoin/rust-secp256k1> for Secp256k1 and <https://github.com/algorand/pairing-plus> for BLS12-381). All other Tier 1 benchmarks are based on BLS12-381 curve in order to make results comparable to Tier 2. Finally, Fig. 13f, shows computation time of Tier 2 for fixed n and various threshold t . As expected, the computation time is linear in n .

References

1. Aranha, D.F., Gouvêa, C.P.L., Markmann, T., Wahby, R.S., Liao, K.: RELIC is an efficient library for cryptography. <https://github.com/relic-toolkit/relic>
2. Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: Craft: composable randomness beacons and output-independent abort MPC from time. Cryptology ePrint Archive, Report 2020/784 (2020). <https://eprint.iacr.org/2020/784>
3. Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: TARDIS: a foundation of time-lock puzzles in UC. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12698, pp. 429–459. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77883-5_15
4. Bhat, A., Shrestha, N., Luo, Z., Kate, A., Nayak, K.: RandPiper - reconfiguration-friendly random beacons with quadratic communication. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021, pp. 3502–3524. ACM Press, November 2021
5. Boneh, D., Boneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 757–788. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_25
6. Boneh, D., Naor, M.: Timed commitments. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 236–254. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_15
7. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)
8. Cascudo, I., David, B.: SCRAPE: scalable randomness attested by public entities. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) ACNS 2017. LNCS, vol. 10355, pp. 537–556. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61204-1_27
9. Cascudo, I., David, B.: ALBATROSS: publicly Attestable BATched randomness based on secret sharing. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12493, pp. 311–341. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64840-4_11

10. Cascudo, I., David, B., Shlomovits, O., Varlakov, D.: Mt. random: multi-tiered randomness beacons. *Cryptology ePrint Archive, Report 2021/1096* (2021). <https://eprint.iacr.org/2021/1096>
11. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) *CRYPTO 1992*. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_7
12. Chen, J., Micali, S.: Algorand: a secure and efficient distributed ledger. *Theor. Comput. Sci.* **777**, 155–183 (2019)
13. Cherniaeva, A., Shirobokov, I., Shlomovits, O.: Homomorphic encryption random beacon. *Cryptology ePrint Archive, Report 2019/1320* (2019). <https://eprint.iacr.org/2019/1320>
14. Cortier, V., Galindo, D., Glondu, S., Izabachène, M.: Distributed elgamal à la pedersen: application to Helios. In: *Proceedings of the 12th Annual ACM Workshop on Privacy in the Electronic Society, WPES 2013*, pp. 131–142. ACM (2013)
15. Daian, P., Pass, R., Shi, E.: Snow White: robustly reconfigurable consensus and applications to provably secure proof of stake. In: Goldberg, I., Moore, T. (eds.) *FC 2019*. LNCS, vol. 11598, pp. 23–41. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32101-7_2
16. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros Praos: an adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) *EUROCRYPT 2018*. LNCS, vol. 10821, pp. 66–98. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_3
17. De Feo, L., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from super-singular isogenies and pairings. In: Galbraith, S.D., Moriai, S. (eds.) *ASIACRYPT 2019*. LNCS, vol. 11921, pp. 248–277. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_10
18. Fouque, P.-A., Stern, J.: One round threshold discrete-log key generation without private channels. In: Kim, K. (ed.) *PKC 2001*. LNCS, vol. 1992, pp. 300–316. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_22
19. Freitag, C., Komargodski, I., Pass, R., Sirkin, N.: Non-malleable time-lock puzzles and applications. In: Nissim, K., Waters, B. (eds.) *TCC 2021*. LNCS, vol. 13044, pp. 447–479. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90456-2_15
20. Galindo, D., Liu, J., Ordean, M., Wong, J.-M.: Fully distributed verifiable random functions and their application to decentralised random beacons. In: *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, 6–10 September 2021*, pp. 88–102. IEEE (2021)
21. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: Stern, J. (ed.) *EUROCRYPT 1999*. LNCS, vol. 1592, pp. 295–310. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_21
22. Gurkan, K., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Aggregatable distributed key generation. In: Canteaut, A., Standaert, F.-X. (eds.) *EUROCRYPT 2021*. LNCS, vol. 12696, pp. 147–176. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_6
23. Hanke, T., Movahedi, M., Williams, D.: Dfinity technology overview series, consensus system (2018)
24. Heidarvand, S., Villar, J.L.: Public verifiability from pairings in secret sharing schemes. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) *SAC 2008*. LNCS, vol. 5381, pp. 294–308. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04159-4_19

25. Katz, J., Loss, J., Xu, J.: On the security of time-lock puzzles and timed commitments. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12552, pp. 390–413. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64381-2_14
26. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
27. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th FOCS, pp. 120–130. IEEE Computer Society Press, October 1999
28. Pedersen, T.P.: A threshold cryptosystem without a trusted party (extended abstract) (rump session). In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 522–526. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_47
29. Pietrzak, K.: Simple verifiable delay functions. In: Blum, A. (ed.) ITCS 2019, vol. 124, pp. 60:1–60:15. LIPIcs, January 2019
30. randao.org. RANDAO: Verifiable random number generation (2017). https://www.randao.org/whitepaper/Randao.v0.85_en.pdf. Accessed 20 Feb 2020
31. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
32. Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., Weippl, E.R.: RandRunner: distributed randomness from trapdoor VDFs with strong uniqueness. In: NDSS 2021. The Internet Society, February 2021
33. Schindler, P., Judmayer, A., Stifter, N., Weippl, E.R.: HydRand: efficient continuous distributed randomness. In: 2020 IEEE Symposium on Security and Privacy, pp. 73–89. IEEE Computer Society Press, May 2020
34. Schoenmakers, B.: A simple publicly verifiable secret sharing scheme and its application to electronic voting. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 148–164. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_10
35. Syta, E., et al.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy, pp. 444–460. IEEE Computer Society Press, May 2017
36. DRAND Team: DRAND project website (2020). <https://drand.love>. Accessed 21 Mar 2021
37. Wang, G., Shi, Z.J., Nixon, M., Han, S.: SoK: sharding on blockchain. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, 21–23 October 2019, pp. 41–61. ACM (2019)
38. Wesolowski, B.: Efficient verifiable delay functions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 379–407. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_13



Revisiting Transaction Ledger Robustness in the Miner Extractable Value Era

Fredrik Kamphuis^{1(✉)}, Bernardo Magri², Ricky Lamberty¹,
and Sebastian Faust³

¹ Corporate Research, Robert Bosch GmbH, Renningen, Germany

{fredrik.kamphuis,ricky.lamberty}@de.bosch.com

² The University of Manchester, Manchester, UK

bernardo.magri@manchester.ac.uk

³ Technical University of Darmstadt, Darmstadt, Germany

sebastian.faust@cs.tu-darmstadt.de

Abstract. In public transaction ledgers such as Bitcoin and Ethereum, it is generally assumed that miners do not have any preference on the contents of the transactions they include, such that miners eventually include all transactions they receive. However, Daian *et al.* S&P'20 showed that in practice this is not the case, and the so called *miner extractable value* can dramatically increase miners' profit by re-ordering, delaying or even suppressing transactions. Consequently an "unpopular" transaction might never be included in the ledger if miners decide to *suppress* it, making, e.g., the standard liveness property of transaction ledgers (Garay *et al.* Eurocrypt'15) impossible to be guaranteed in this setting.

In this work, we formally define the setting where miners of a transaction ledger are dictatorial, i.e., their transaction selection and ordering process is driven by their individual preferences on the transaction's contents. To this end, we integrate dictatorial miners into the transaction ledger model of Garay *et al.* by replacing honest miners with dictatorial ones. Next, we introduce a new property for a transaction ledger protocol that we call *content preference robustness* (CPR). This property ensures *rational liveness*, which guarantees inclusion of transactions even when miners are dictatorial, and it provides *rational transaction order preservation* which ensures that no dictatorial miner can improve its utility by altering the order of received candidate transactions. We show that a transaction ledger protocol can achieve CPR if miners cannot obtain a-priori knowledge of the content of the transactions. Finally, we provide a generic compiler based on time-lock puzzles that transforms any robust transaction ledger protocol into a CPR ledger.

Keywords: blockchain · liveness · rational security

1 Introduction

In distributed transaction ledgers such as Bitcoin [26] and Ethereum [8], transactions proposed by users are verified in a decentralized way and appended into

a public ledger in an unalterable order. To this end, a set of network participants called *miners* are responsible for the process of including and finalizing transactions by running the consensus protocol. The *liveness* property of a consensus protocol guarantees that a correctly generated transaction that is provided as input to all the miners will eventually appear on the ledger. It has been formally shown that this guarantee is achieved under the assumption of honest majority (or supermajority) of miners [3, 16, 25]. Therefore, it is commonly assumed that honest miners input all the transactions to the consensus protocol in the exact same order they were received. In practice, transaction ledger protocols usually establish incentive mechanisms (e.g., in the form of transaction fees) to justify this fundamental assumption. A transaction ledger protocol therefore aims to achieve self-enforced *honest behavior* by incentivizing parties to behave honestly and penalizing deviation of the desired protocol behavior [4, 21]. These mechanisms yield profit for miners, e.g. for honestly including and appending transactions into blocks. Nevertheless, when analyzing the incentive compatibleness of *honest behavior*, the approach usually taken [2] is based on the assumption that miners do not have any rational interest in the actual content of the transactions they include. As it turns out, this assumption cannot be guaranteed in practice. In fact, there are many works that show that rational miners indeed profit from altering the order or even ignoring certain transactions entirely [10, 15, 30, 31]. While forking on the underlying ledger to revert transactions could theoretically lead to the same results, it requires at least 1/3rd or the majority of the resources of the network (i.e., computation or stake) to be executed [7, 22, 24]. Additionally, such an attack could lead to unforeseeable dynamics that might be undesirable for rational adversaries [2, 7]. Moreover, reordering, delaying or suppressing single transactions during the inclusion in the block is a much more subtle deviation from the honest behavior compared to forking, and more importantly can be accomplished by any individual miner. Therefore, this type of behavior might be considered as a viable and practical strategy by rational miners [10, 27].

Daian *et al.* [10] generalizes this concept of content-depending utilities as *miner extractable value* (MEV). MEV is a metric representing all kinds of opportunities a rational miner can generate utility permissionlessly from e.g., re-ordering, delaying or censoring of transactions depending on the transaction's content. [10] shows that MEV is not just a theoretical concept, but rather a real-world phenomenon that is already occurring at scale in today's DeFi¹ space, e.g. in the form of transaction front-running, reaching its current spike with roughly 25000 ETH (4.1 million USD) available for arbitrage daily². Overall, the actual content preference of rational miners depends on on-chain dynamics within the transaction ledger protocol, such as arbitrage opportunities, front running and censorship, but also on utility sources outside the ledger. This might include cross-chain dynamics where miners can generate profit on other chains by tak-

¹ The term *decentralized finance* (DeFi) refers to an alternative financial infrastructure, that is built on top of open and permissionless protocols, such as the Ethereum blockchain.

² Flashbots. <http://explore.flashbots.net/>, as of July 09, 2022.

ing rational actions on their chain [24], but also off-chain dynamics where the profit for taking rational actions is generated entirely off-chain [7]. Therefore, the actual individual utility a rational miner can generate for taking actions against candidate transactions is unknown to the public and the protocol designer.

Further, as illustrated by Daian *et al.* unclaimed MEV opportunities in a branch of the blockchain can incentivize miners to support that branch and abandon the longest chain, thus “rolling back” blocks and creating potential forks. This harms the network and even poses a fundamental threat to the security of the underlying consensus protocol.

To summarize, the inherent issue is that miners can use their a-priori knowledge of the transaction content to alter the set of transactions they are supposed to include. This information asymmetry provides miners with an advantage by taking rational actions (e.g., altering order, delaying, suppressing) which may be rewarded disproportionately in comparison to honest behavior [10, 15]. Moreover, these rational actions are not a violation of the underlying transaction ledger protocol, and thus not captured by existing security definitions. The miners that are willing to take rational actions based on the transactions’ contents are referred to as *dictatorial* miners.

Due to space limitations the related work section is presented in Appendix A.

1.1 Contributions

In this work we analyze the liveness and transaction order guarantee that can be achieved against dictatorial miners. In this vein, we follow the Bitcoin backbone model of Garay, Kiayias and Leonardos [16] with the twist that instead of honest miners we assume a set of dictatorial miners with hidden³ preferences over the contents of transactions. The dictatorial miners participate honestly in the consensus protocol but may choose to take rational actions that do not violate the properties of the ledger protocol, e.g. re-order, delay or suppress a particular set of transactions. In addition, dictatorial miners are allowed to collude with each other if it is (individually) more profitable. Our contributions can be summarized as follows:

- We introduce a new property for transaction ledger protocols that we call *content preference robustness* (CPR), which yields robustness against dictatorial miners. A CPR-ledger provides the following guarantees:
 - *Rational liveness*: It provides essentially the same guarantees as the original liveness definition [16], but against dictatorial miners. To achieve this we show that no dictatorial miner can increase its profit by selectively suppressing transactions if (1) dictatorial miners gain no a-priori knowledge of the transactions contents, (2) withholding transactions is punishable by the ledger and (3) dictatorial miners expect to not lose utility (on average) by honestly participating in the transaction ledger protocol.

³ We call *hidden* the preferences that are individual to the miners, and not known to the protocol designer.

- *Rational transaction order preservation*: It ensures that no dictatorial miner can improve its expected utility by altering the order of received transactions. We show that rational transaction order preservation can be achieved under essentially the same conditions as rational liveness.
- We present a generic compiler based on time-lock puzzles that compiles any robust transaction ledger protocol (according to [16]) into a transaction ledger protocol that is CPR. On a technical level, the compiled protocol maintains two (logically) separate chains; the “control chain” that contains only time-lock puzzles of the transactions, and the “sanitized chain” that contains the actual contents of the transactions from (the common prefix of) the control chain. The intuition is that the control chain provides a global ordering of the transactions for all miners, and once that ordering is fixed, the sanitized chain can be built with the actual contents of all valid transactions from the control chain. Finally, we show that the compiled protocol is CPR.

2 Transaction Ledger Model

In this work we extend the transaction ledger model of Garay *et al.* [16] to the setting where dictatorial miners may use their a-priori knowledge of the transaction content in order to generate MEV. To this end, we first state the fundamentals of the transaction ledger model by Garay *et al.* [16] and then continue to explain how we adapt their model.

2.1 Ledger Backbone Model

According to Garay *et al.* [16] a transaction ledger is represented as a vector of blocks $l = (\mathcal{B}_1, \dots, \mathcal{B}_d)$, where each block $\mathcal{B}_i = (tx_1, \dots, tx_n)$ is a vector of transactions $tx \in \mathcal{T}$. \mathcal{T} denotes the set of valid transactions. Appending a transaction tx to a vector l is denoted by $l||tx$. Also, appending a vector of transactions \mathcal{B} to another vector l is denoted as $l||\mathcal{B}$. $tx_{i,j}$ denotes transaction tx_j in block \mathcal{B}_i . As a ledger is a vector of transactions, we simply denote it as $l = (tx_1, \dots, tx_m)$ omitting the block numbers when clear from the context.

The transaction ledger protocol is executed by a set of miners \mathcal{M} in the presence of a PPT adversary \mathcal{S} , and driven by a PPT environment \mathcal{Z} . Each honest miner M_i maintains its own local copy of the chain l_i . Further, an honest miner M_i process a local buffer $X_i := (tx_1, \dots, tx_e)$, that are candidate transactions to be incorporated into the ledger l_i provided by the environment \mathcal{Z} .⁴ In [16], a transaction ledger protocol is defined by the transaction generation oracle TxGen that issues transactions on behalf of the users and the set of valid ledgers \mathcal{L} . Upon receiving a message (IssueTx, γ , P), TxGen generates a unique transaction $tx[\gamma] \in \mathcal{T}$ on behalf of P , where $tx[\gamma]$ denotes a transaction tx that contains an encoding of content γ . Further, a ledger is defined by the three interface functions $V(\cdot)$, $I(\cdot)$, $R(\cdot)$. Where $V(\cdot)$ is the chain validity function, $I(\cdot)$ is the

⁴ Note that honest miners operate X_i as provided by the environment and do not change any ordering to maximize fees.

input contribution function that is executed to provide new blocks, and $R(\cdot)$ is the chain reading function that returns a semantic interpretation of the ledger.

Moreover, a transaction ledger protocol is called *robust* if it provides *persistence* and *liveness*. Persistence means that a transaction that appears ‘deep enough’ into the chain will appear at the same position for all honest miners. On the other hand, liveness means that if a transaction is input to the honest miners for at least v rounds it will appear k blocks deep into the chain of those miners.

In our work we assume the existence of a robust transactions ledger protocol $\Pi = (I, V, R, \text{TxGen}, \mathcal{L})$ for some liveness parameter v . For more details on the ledger backbone protocol we refer the reader to the Appendix B.

2.2 Dictatorial Miners

In our model, the transaction ledger protocol is executed by miners in the presence of a PPT adversary \mathcal{S} , and driven by a PPT environment \mathcal{Z} . The adversary \mathcal{S} can fully corrupt a minority of the miners (as in [16]). In contrast to the model of [16], every miner that is not fully corrupt will automatically be dictatorial, leaving no honest miner in the protocol. The difference between an honest miner and a dictatorial miner is that a dictatorial miner has preferences over the content of transactions and might therefore re-order, delay, or suppress transactions it receives from the environment.

Formally, a dictatorial miner M_i receives, at the beginning of each round, in its transaction buffer X_i candidate transactions provided by the environment \mathcal{Z} , just as honest miners would. However, M_i may execute the input contribution function $I(\cdot)$ on a modified X'_i instead. At the beginning of each round M_i may choose X'_i depending on the received transaction buffer X_i and the current ledger l . For every transaction $tx \in X_i$, M_i decides whether to *include* tx in X'_i , to *suppress* it, or to *delay* it to a later round.

When deciding on how to treat a candidate transaction $tx[\gamma]$ a dictatorial miner is assumed to maximize its expected utility with respect to its private utility function $u_i : l \times \Gamma \mapsto \mathbb{R}$. The utility function $u_i(l, \gamma)$ computes the utility of M_i for including a transaction $tx[\gamma]$ into the ledger ⁵. If a miner M_i includes a transaction tx without knowing its content the expected utility for including this transaction is denoted as $u_i(l, tx)$. The miner’s expectation is taken over the distribution on the transaction’s contents supplied by the environment \mathcal{Z} which is assumed to be common prior for all miners⁶.

⁵ The utility function covers all kinds of revenues a miner might expect including fees, extractable values, bribes. Since, estimating this utility is rather complex we assume the function only to be known to the respective miner itself.

⁶ For the simplicity of our model we assume that all dictatorial miners share a common belief on the contents of transactions the environment \mathcal{Z} will provide. In practice, this belief might be different for the miners considering the information available to the miners. However, aligning the beliefs might be part of the collaboration between the miners.

We say that a miner prefers to include a transaction $tx_0[\gamma_0]$ over transaction $tx_1[\gamma_1]$ in some ledger l if $u_i(l, \gamma_0) > u_i(l, \gamma_1)$. The preference is denoted as $tx_0[\gamma_0] \succ tx_1[\gamma_1]$ if the ledger l is evident from the content. A miner is indifferent between $tx_0[\gamma_0]$ and $tx_1[\gamma_1]$, denoted as $tx_0[\gamma_0] \sim tx_1[\gamma_1]$, if $u_i(l, \gamma_0) = u_i(l, \gamma_1)$. Appending a new transaction results in a new ledger, therefore the utility for appending $tx_0[\gamma_0]$ and then appending $tx_1[\gamma_1]$ is $u_i(l, \gamma_0) + u_i(l|\gamma_0, \gamma_1)$. Finally, we denote an empty content of a transaction as \perp . The expected utility for including an empty or neutral content \perp is assumed to be $u_i(l, \perp) = 0$, since including an empty transaction results semantically in the same ledger.⁷

Miner's Coalition. Dictatorial miners may collude with each other if forming a coalition yields a higher individual utility. In fact, [10] shows that even if miners have concurrent preferences, e.g., they are concurring for the same MEV opportunity, collaborative behavior is not just stable but in fact yields a higher individual utility. Therefore, it is assumed (and observed in practice) that dictatorial miners will eventually collude if it is individually more profitable [10]. Our model allows dictatorial miners to coordinate their actions against single transactions, thus whenever there exists an individually profitable coalition of dictatorial miners, one could see it as a single entity.⁸ Note that this may include the grand coalition of all miners.

Forking. We stress, that dictatorial miners execute the same input contribution function as honest miners (with potentially different inputs), thus only creating valid blocks and extending the current chain. While a sufficiently large coalition of malicious miners could fork the the longest chain, the simple reorder, delay or suppression of transactions that can be performed by dictatorial miners is a much more subtle deviation from the honest behavior compared, thus allowing for a positive result even against a coalition of all dictatorial miners.

3 Content Preference Robustness

In this section we formalize the concept of content preference robustness, that yields liveness and transaction order guarantee against dictatorial miners. To this end, we define the properties *rational liveness* and *rational transaction ordering*. Further, we explain the rationale of restricting dictatorial miners beliefs in order to exclude trivial cases from our model.

3.1 Rational Liveness

Rational liveness says that a transaction that is input to all dictatorial miners will eventually appear in the ledger. Consider a transaction ledger protocol $\Pi = (I, V, R, \text{TxGen}, \mathcal{L})$. Rational liveness is defined as:

⁷ Appending an empty transaction does not change, e.g., any balances nor account states.

⁸ Practically, this means that dictatorial miners might even exchange secret information, e.g. if a transaction content is secret shared amongst the miners.

Definition 1 (Rational liveness). *If a transaction $tx[\gamma] \in \mathcal{T}$ issued by $\text{T}\times\text{Gen}$ is input for all dictatorial miners for at least v consecutive rounds, then all dictatorial miners will report this transaction at least k blocks deep into the ledger, for some $k, v \in \mathbb{N}$.*

Intuitively, rational liveness provides the same guarantees as the liveness definition of [16], but extended to dictatorial miners. In order to show that a *robust* transaction ledger protocol Π achieves this property one has to show that it is in the dictatorial miners best interest to behave as an honest miner, such that the transactions provided by the environment are forwarded unchanged to the input contribution function.

3.2 Rational Transaction Ordering

The original model of [16] does not formalize the concept of transaction ordering. However, as practically demonstrated by [10] dictatorial miners can extract significant utility by rearranging transactions in different ways. Therefore, transaction ordering is of major relevance when considering dictatorial behavior of the miners. To this end we define rational transaction ordering preservation as follows:

Definition 2 (Rational transaction ordering). *A transaction ledger protocol $\Pi = (\mathbb{I}, \mathbb{V}, \mathbb{R}, \text{T}\times\text{Gen}, \mathcal{L})$ preserves rational transaction ordering if for all pairs of transactions (tx_0, tx_1) issued by $\text{T}\times\text{Gen}$, for all ledgers $l \in \mathcal{L}$ and for all miners $M_i \in \mathcal{M}$, we have that $u_i(l||tx_0, tx_1) = u_i(l||tx_1, tx_0)$.*

Intuitively, rational transaction order preservation means that a dictatorial miner receives the same expected utility for including transaction tx_0 before transaction tx_1 into a ledger l for all pairs (tx_0, tx_1) and all ledgers $l \in \mathcal{L}$. In particular this means that a dictatorial miner does not improve its expected utility by altering the order of transactions received by the environment, and hence has no incentive to do so.

3.3 Restrictions on the Environment

Since we allow the miner's utility function to depend on off-chain dynamics, it is possible that the utility of a miner is always negative. Thus, this miner would do strictly better by always suppressing all transactions, independently of their content (or the current ledger). This might be induced, e.g., by some utility source that offers a large compensation for mining empty blocks. Since this kind of utility source would make the rational decision independently of the transaction's contents or the ledger state, we explicitly exclude them from our model.

For this, we limit the expectations of the miners about the environment \mathcal{Z} such that all transactions sent to the miners are expected to yield at least a positive utility, as otherwise the miners are not incentivized to include transactions.

Definition 3 (Expected incentive compatible environment). *An environment \mathcal{Z} providing transactions to a set of miners \mathcal{M} executing the ledger protocol Π is expected incentive compatible if for all miners $M_i \in \mathcal{M}$, for all ledgers $l \in \mathcal{L}$, and for all $tx \in \mathcal{T}$ we have that $u_i(l, tx) > 0$.*

Note, that this does not restrict the environment itself but rather the believe of the dictatorial miners about the environment. Intuitively, this means that the miners expect the environment to provide transactions that yield a positive utility on average, where the expectation is taken over the distribution of transaction contents the environment provides. This assumption is essential to ensure that dictatorial miners will eventually improve their expected utility by including transactions. Note that miners expecting to lose utility for including transactions provided by the environment would trivially not include it. In practice, this is usually accomplished by transaction fee mechanisms.⁹ Note however that this assumption *does not* trivialise the problem, as even with an expected incentive compatible environment, dictatorial miners could still increase their profits by, e.g., suppressing or reordering selected transactions from the ledger.

Content Preference Robustness. Putting all together, we now define the notion of *content preference robustness* (CPR) for a transaction ledger protocol executed in the presence of dictatorial miners.

Definition 4 (content preference robustness). *A transaction ledger protocol $\Pi = (I, V, R, \text{TxGen}, \mathcal{L})$ executed by a set of dictatorial miners \mathcal{M} driven by some expected incentive compatible environment \mathcal{Z} in presence of a PPT adversary \mathcal{S} is called CPR if Π achieves rational liveness and rational transaction ordering.*

4 Compiling a Robust Ledger into a CPR Ledger

In this section we show how to get *content preference robustness* for a transaction ledger protocol. In particular, we show how to generically compile a *robust* transaction ledger protocol into a CPR transaction ledger protocol.

4.1 CPR Compiler

A CPR compiler for a robust transaction ledger protocol Π is defined as follows.

Definition 5 (CPR Compiler). *Let $\Pi = (I, V, R, \text{TxGen}, \mathcal{L})$ be a robust transaction ledger protocol. A CPR compiler Φ is a PPT algorithm $\Pi' \leftarrow \Phi(\Pi)$ such that $\Pi' = (I', V', R', \text{TxGen}', \mathcal{L}')$ achieves CPR.*

In the following we provide an overview of our CPR compiler, followed by a detailed description of how our CPR-compiler Φ transforms each component of a robust transaction ledger protocol Π to build the CPR ledger protocol Π' .

⁹ Note, that in practice it would be sufficient for the dictatorial miners to believe that the environment is incentive compatible for themselves. However, for the sake of simplicity of our model we assume that miners believe that the environment is expected incentive compatible for every miner.

4.2 Compiler Overview

The CPR compiler modifies the transaction generation oracle TxGen into a “time-lock transaction generation oracle”, that issues transactions inside time-lock puzzles [5, 28] that can be opened after some specified time has passed. The idea is to let the miners commit to a set and order of time-locked transactions before their content gets revealed. This forces the miners to make decisions about incoming transactions before knowing the actual content of the transactions. However, the miners might try to delay transactions until their content gets revealed before making their final decision. Therefore, the protocol II' has to ensure that delaying transactions is not expected to be profitable for the miners.

The compiled transaction ledger protocol II' maintains two separate chains; the “control” chain l'_c that contains a ledger consisting of time-locked transactions, and the “sanitized” chain l'_m that contains the actual contents of the transactions that are final in the control chain. The mining of new blocks follows the rules of the underlying robust ledger protocol II (e.g., Proof-of-Work), and it occurs in the control chain first. To extend the ledger, miners first gather all the new (time-locked) transactions from their input buffer and build a block. The miner that wins the right to append a block to the control chain is also responsible for extending the sanitized chain by providing solutions to the time-lock puzzles that are in the common-prefix of the control chain.

To illustrate with a concrete example, consider a robust ledger protocol with liveness parameter v of 2 rounds, i.e., after 2 rounds a transaction is considered final in the ledger, e.g. being $k = 2$ blocks deep into the chain. Assume a time-lock puzzle instantiation that can only be opened 2 rounds after its creation. Hence, for the first 2 rounds of the protocol only the control chain will be extended, as no solution to puzzles is yet available. At round 3 the miner that creates the new block on the control chain must also create a new block on the sanitized chain by providing solutions to all the puzzles that are included in the block created at round 1 in the control chain; the solution to the puzzles are in fact the contents of the transactions. From round 3 onwards, every new round will extend both chains, and the sanitized chain at round r will contain the contents of the transactions included at round $r - 2$ in the control chain. As all transactions that are at least 2 blocks deep in the ledger are final, then the blocks in the sanitized chain only includes contents that are already final. The structure of the build ledger is illustrated in Fig. 1.

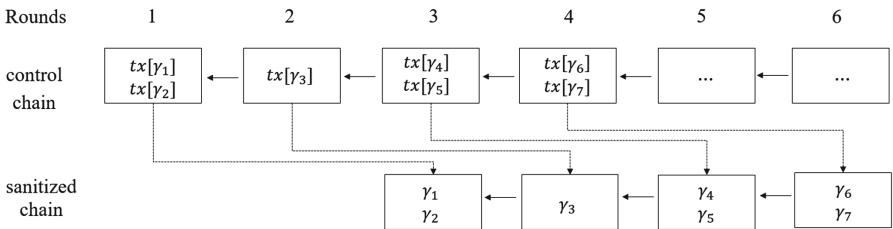


Fig. 1. High-level architecture of a CPR ledger with liveness parameter $v = 2$.

Note that the control chain is the only chain that needs consensus, as the state of the sanitized chain can be deterministically retrieved from the control chain. However, miners can still freely choose the time-locked transactions to be included in the control chain, and therefore try to delay a time-locked transaction until learning its content before deciding to either include it or suppress it. To this end, our construction implements a mechanism that invalidates transactions that are “too old”. Thus, dictatorial miners face the dilemma of either including a time-locked transaction without knowing its content, or delaying it, which leads to the invalidation of the transaction and loss of utility. To solve the dilemma, rational miners must rely on their expectation on the transactions’ content.

4.3 Time-Lock Transaction Generation Oracle

Here we describe how our CPR compiler Φ builds the time-lock transaction generation oracle TxGen' for the compiled protocol Π' . A natural way to model a time-lock transaction generation oracle TxGen' is by describing it as a functionality. The functionality we need from such an oracle is that transactions can be created on behalf of users and the miners are merely notified that a transaction has been created. Moreover, published transactions should be associated with a fresh ledger account. The content corresponding to the transaction can only be retrieved after some predetermined number of rounds.

More formally, the $\mathcal{F}_{\text{tl-TxGen}}$ functionality encapsulates the content γ of a transaction into a randomly generated transaction tag \tilde{tx} that is delivered to every miner. A miner can learn the content associated with a transaction tag *at least* δ rounds after the transaction has been issued by sending at least δ solve requests for the same transaction tag to $\mathcal{F}_{\text{tl-TxGen}}$. Once, the correct amount of requests is issued the functionality returns the associated solution tag sid . Further, the functionality allows to verify if a content belongs to a transaction tag by returning the corresponding content. To this end, on receiving a solution tag sid at any time the functionality $\mathcal{F}_{\text{tl-TxGen}}$ returns the associated content γ . By separating, solving the puzzle from revealing the message it is ensured that a message can be revealed at any time if sid is known, even without solving the puzzle. Note that in our functionality, the issuing round of a transaction as well as a fresh ledger account are also associated with the transaction tag.

Practically, this means that users sign the time-locked transaction using a freshly generated ledger account and include a time stamp of the creation time, which relates to the current round.¹⁰ The first can be used for practical reasons to deter malicious users from flooding the ledger with time-locked transactions, as we will discuss later on in Sect. 5. The latter is used by the protocol to determine whether a transaction is “too old”. The $\mathcal{F}_{\text{tl-TxGen}}$ functionality is described next.

¹⁰ According to Garay *et al.* this would practically relate to the block number. Therefore this “timestamping” of the creation round is practically achieved by including the latest known block number.

Functionality $\mathcal{F}_{\text{tl-TxGen}}$

The $\mathcal{F}_{\text{tl-TxGen}}$ functionality is parameterized by a delay parameter δ that represents how many rounds the transaction content stays hidden, a set of transaction tags \tilde{T} , a set of ledger accounts \mathcal{A} , and space of solution tags Sid . It is executed by a set of users \mathcal{P} , a set of miners \mathcal{M} , and an adversary \mathcal{S} . $\mathcal{F}_{\text{tl-TxGen}}$ maintains initially empty lists \mathcal{O} and \mathcal{Q} .

- On message (**IssueTx**, txid, γ) from $P_i \in \mathcal{P}$ in some round r the functionality samples $\tilde{tx} \leftarrow_{\mathcal{S}} \tilde{T}$, $\text{sid} \leftarrow_{\mathcal{S}} \text{Sid}$, and the account $A \leftarrow_{\mathcal{S}} \mathcal{A}$. Then, it records the tuple $(\text{txid}, \tilde{tx}, \text{sid}, \gamma, r, A, P_i)$ in \mathcal{O} and sends it to P_i . Further, send the message (**Issued**, $\text{txid}, \tilde{tx}, r, A$) to every $M \in \mathcal{M}$ and \mathcal{S} .
- On message (**Solve**, txid, \tilde{tx}) from $M_i \in \mathcal{M}$ in some round r the functionality does the following: If no record $(\text{txid}, \tilde{tx}, r, M_i)$ exists in \mathcal{Q} append $(\text{txid}, \tilde{tx}, r, M_i)$ to \mathcal{Q} . Let L be the set of records of the form $(\text{txid}, \tilde{tx}, \cdot, M_i)$ in \mathcal{Q} and $(\text{txid}, \tilde{tx}, \text{sid}, \gamma, r', A, P_j)$ a record in \mathcal{O} . If $|L| \geq \delta$ send the message $(\text{txid}, \tilde{tx}, \text{sid}, r, M_i)$ to M_i and \mathcal{S} . Otherwise send message $(\text{txid}, \tilde{tx}, \perp, r, M_i)$.
- On message (**RevealMsg**, $\text{txid}, \tilde{tx}, \text{sid}$) from $M_i \in \mathcal{M}$ in some round r the functionality does the following: If there is a record $(\text{txid}, \tilde{tx}, \text{sid}, \gamma, r', A, P_j)$ in \mathcal{O} send the message $(\text{txid}, \tilde{tx}, \text{sid}, \gamma, r', A, P_j, r, M_i)$ to M_i and \mathcal{S} . Otherwise send message $(\text{txid}, \tilde{tx}, \text{sid}, \perp, \perp, \perp, \perp, r, M_i)$.

Note that we intentionally separate the set of users (that only post transactions) and miners (that process the transactions) in the $\mathcal{F}_{\text{tl-TxGen}}$ functionality. This is to better illustrate that our results only concern the case where miners do not send transactions. It is easy to see that whenever a miner itself generates transactions it is not possible to prevent this miner from learning the content of its own transaction. Therefore, we restrict the functionality to only accept **IssueTx** commands from users.

In order to instantiate the transaction generation oracle TxGen' of the compiled protocol Π' the functionality $\mathcal{F}_{\text{tl-TxGen}}$ should be parameterized with a delay parameter $\delta = v$, where v is the liveness parameter of Π . Further, the content space Γ' for the protocol Π' corresponds to the transaction space \mathcal{T} in the support of the transaction generation oracle TxGen of the underlying transaction ledger protocol Π . Intuitively, this means that the environment \mathcal{Z} samples transactions in the support of TxGen that are provided as content to $\mathcal{F}_{\text{tl-TxGen}}$ when issuing a transaction.¹¹

Realizing the $\mathcal{F}_{\text{tl-TxGen}}$ Functionality. Our $\mathcal{F}_{\text{tl-TxGen}}$ functionality is a simplified version of the time-lock puzzle functionality of [5, Fig. 3], where we simply cast it as a transaction generation oracle. In particular, the original functionality sam-

¹¹ Nevertheless, the functionality $\mathcal{F}_{\text{tl-TxGen}}$ is not restricted to a specific content space.

ples a consecutive sequence of puzzle states, while sending a solve request returns the next state in the sequence. For the sake of simplicity of our functionality we omit the intermediary states. Instead in our functionality a puzzle tag is queried multiple times to solve the puzzle until the functionality returns a solution tag. This solution tag corresponds to the last puzzle state from the original functionality and can be used to reveal the message at any time as in [5]. Note, that this simplification does not interfere with the security since no additional information is leaked. Additionally, our functionality samples a new account that is associated with the transaction tag. Since this tag is sampled uniformly at random it does not leak any information about the content or solution tag associated with the transaction tag and therefore does not have any impact on the security.

In [5], it was shown that (a version of) the well known time-lock puzzle construction by Rivest, Shamir and Wagner [28] realizes the time-lock puzzle functionality of [5] in the Universal Composition (UC) model [9]. The time-lock puzzle construction of [28] is based on the assumption that it is hard to compute repeated squarings of an element of $(\mathbb{Z}/N\mathbb{Z})^\times$ with large N in less time than it would take to compute each of the squarings sequentially, unless the factorization of N is known. Therefore, in order to solve a time lock puzzle a miner has to perform a predefined amount sequential squarings.

Hence, by the composition property of the UC framework, we can simply use the time-lock puzzle protocol of [5] as a plug-in replacement for our functionality.

4.4 Chain Validity Function

A ledger l' of the compiled transaction ledger protocol Π' consists of the control chain l'_c and the sanitized chain l'_m . The control chain is a ledger of tuples $tx' = (\text{txid}, \tilde{tx}, A)$ providing time-locked transaction tags \tilde{tx} from the tag space \tilde{T} , an associated ledger account A from the the ledger account space \mathcal{A} , and unique transaction identifiers txid. The sanitized chain l'_m is a ledger of transaction contents containing tuples of the form $\gamma' = (\text{sid}, \gamma, r, P)$. A transaction ledger l' consists of blocks $(\mathcal{B}_1^{r_1}, \dots, \mathcal{B}_n^{r_n})$ where for each block $\mathcal{B}_i^{r_i}$, r_i denotes the round the block is created in. Each block extends the control chain l'_c and the sanitized chain l'_m . Therefore, each block is a tuple $\mathcal{B}_i^{r_i} = (\mathcal{B}_{c,i}^{r_i}, \mathcal{B}_{m,i}^{r_i})$, with $\mathcal{B}_{c,i}^{r_i} = (tx'_{i,1}, \dots, tx'_{i,y})$ and $\mathcal{B}_{m,i}^{r_i} = (\gamma'_{i,1}, \dots, \gamma'_{i,q})$. For simplicity we denote $l' = (l'_c, l'_m) = ((tx'_1, \dots, tx'_p), (\gamma'_1, \dots, \gamma'_q))$ when referring to the concatenation of all blocks. A ledger l' is in the set of valid ledgers \mathcal{L}' if the chain validation function V' returns *true* on l' . Intuitively, $V'(\cdot)$ checks for every content in the sanitized chain if it corresponds to the transaction tag at the same position in the control chain. The formal algorithm for checking if a ledger l' is a valid ledger for the transaction ledger protocol Π' is provided in Algorithm 1.

4.5 Input Contribution Function

The input contribution function $l'(\cdot)$ is executed in order to generate an updated ledger. It receives as input a current transaction ledger $l' = (l'_c, l'_m) = ((tx'_1, \dots, tx'_p), (\gamma'_1, \dots, \gamma'_q))$, a buffer of not included transactions X . Further, $l'(\cdot)$

Algorithm 1. $V'(l')$

```

1: parse  $(l'_c, l'_m) \leftarrow l'$ 
2: parse  $(tx'_1, \dots, tx'_p) \leftarrow l'_c$ 
3: parse  $(\gamma'_1, \dots, \gamma'_q) \leftarrow l'_m$ 
4: for  $i = 1, \dots, q$  do
5:   parse  $(txid_i, \tilde{tx}_i, A) \leftarrow tx'_i$ 
6:   parse  $(sid_i, \gamma_i, r_i, P_i) \leftarrow \gamma'_i$ 
7:   send  $(\text{RevealMsg}, txid_i, \tilde{tx}_i, sid_i)$  to
       $\mathcal{F}_{\text{tl-TxGen}}^\delta$  to receive  $(txid_i, \tilde{tx}_i, sid_i, \tilde{\gamma}_i, \tilde{r}_i, P_i)$ 
8:   if  $\tilde{\gamma}_i \neq \gamma_i \vee \tilde{r}_i \neq r_i \vee \tilde{P}_i \neq P_i$  then:
9:     return false
10:  end if
11: end for
12: return true

```

is stateful by maintaining a buffer of time-lock puzzle solutions C . Intuitively, $l(\cdot)$ starts with solving all transaction tags from the control chain that are not solved yet. All found solutions are stored in the solution buffer. Then, in order to extend the ledger l' the function $l'(\cdot)$ extends the sanitized chain with the contents of transaction tags from the block that is k blocks deep into the control chain. Further, $l'(\cdot)$ extends the control chain by selecting a set of new time-lock transaction tags from X . Finally, $l'(\cdot)$ includes for all selected time-locked transactions the current round number.¹² The formal algorithm is shown in Algorithm 2.

Algorithm 2. $l'(l', X, r)$

```

1: parse  $(l'_c, l'_m) \leftarrow l'$ 
2: parse  $(tx'_1, \dots, tx'_p) \leftarrow l'_c$ 
3: parse  $(\gamma'_1, \dots, \gamma'_q) \leftarrow l'_m$ 
4: for  $i = q + 1, \dots, p$  do
5:   parse  $(txid_i, tx_i, A) \leftarrow tx'_i$ 
6:   send  $(\text{Solve}, txid_i, \tilde{tx}_i)$  to  $\mathcal{F}_{\text{tl-TxGen}}^\delta$ 
      to receive message  $(txid_i, \tilde{tx}_i, sid_i)$ 
7:   if  $sid_i \neq \perp$  then:
8:     send  $(\text{RevealMsg}, txid_i, \tilde{tx}_i, sid_i)$ 
      to  $\mathcal{F}_{\text{tl-TxGen}}^\delta$  to receive  $(txid_i, \tilde{tx}_i, sid_i, \gamma_i, r_i, A, P_i)$ 
9:     append  $(txid_i, \tilde{tx}_i, sid_i, \gamma_i, r_i, A, P_i)$  to  $C$ 
10:  end if
11: end for
12: parse  $(\mathcal{B}_{c,1}^{r_1}, \dots, \mathcal{B}_{c,n}^{r_n}) \leftarrow l'_c$ 
13:  $j \leftarrow n + 1 - k$ 
14: parse  $(tx'_{q+1}, \dots, tx'_{q+y}) \leftarrow \mathcal{B}_{c,j}^{r_j}$ 
15:  $\mathcal{B}_{m,n+1}^r = \perp$ 
16: for  $i = q + 1, \dots, q + y$  do
17:   get  $(txid_i, \tilde{tx}_i, sid_i, \gamma_i, r_i, A, P_i)$  from  $C$ 
18:    $\gamma' \leftarrow (sid_i, \gamma_i, r_i, P_i)$ 
19:    $\mathcal{B}_{m,n+1}^r = \mathcal{B}_{m,n+1}^r || \gamma'$ 
20: end for
21: get  $(tx'_p, \dots, tx'_{p+j})$  from  $X$ 
22:  $\mathcal{B}_{c,n+1}^r \leftarrow (tx'_p, \dots, tx'_{p+j})$ 
23:  $\mathcal{B}_{n+1}^r = (\mathcal{B}_{c,n+1}^r, \mathcal{B}_{m,n+1}^r)$ 
24: return  $l' = l' || \mathcal{B}_{n+1}^r$ 

```

Remark. One could also separate the tasks of solving time-lock puzzles and extending the chain into different functions. However, since extending the ledger depends on solving the puzzles we simply incorporate solving the puzzles into $l(\cdot)$. By doing so, we additionally stick closer to the model of [16].

¹² According to Garay *et al.* this “timestamping” of the inclusion round is practically achieved by giving a block number to the selected transactions.

4.6 Chain Reading Function

The chain reading function $R'(\cdot)$ returns a semantic interpretation of a ledger $l' \in \mathcal{L}'$. It receives as input a transaction ledger l' and internally calls the chain validation function $V(\cdot)$ and chain reading function $R(\cdot)$ of the underlying transaction ledger protocol Π . The intuition is that the revealed contents in the sanitized chain contain transactions in the support of $\text{T}\times\text{Gen}$ from the protocol Π . Therefore, $R'(\cdot)$ can determine the longest chain that is a valid ledger with respect to $V(\cdot)$. This longest chain can then be interpreted by $R(\cdot)$. Additionally, $R'(\cdot)$ checks for every transaction content in the sanitized chain if the corresponding time-locked transaction tag was included “too late” in the control chain by comparing the revealed round number of the transaction tag generation with the round number of the block that included the transaction tag. If the difference between block creation round and transaction creation round is more than the liveness parameter v the content gets ignored. The function $R'(\cdot)$ ensures that transaction contents that are considered as “too old” are not interpreted by $R(\cdot)$ and are therefore not considered for the semantic interpretation of the ledger l' . The algorithm for R' is shown in Algorithm 3.

Algorithm 3. $R'(l')$

```

1: if  $V(l') = \text{false}$  then:
2:   return  $\perp$ 
3: end if
4:  $(l'_c, l'_m) \leftarrow l'$ 
5: parse  $(\mathcal{B}_{c,1}^{r_1}, \dots, \mathcal{B}_{c,n}^{r_n}) \leftarrow l'_c$ 
6: parse  $(\mathcal{B}_{m,1}^{r_1}, \dots, \mathcal{B}_{m,n}^{r_n}) \leftarrow l'_m$ 
7:  $\tilde{l} \leftarrow \perp$ 
8: for  $i = 1, \dots, n$  do
9:    $(\gamma'_{i,1}, \dots, \gamma'_{i,y}) \leftarrow \mathcal{B}_{m,i}^{r_i}$ 
10:  for  $j = 1, \dots, y$  do
11:    parse  $(\text{sid}_j, \gamma_j, r_j, P_j) \leftarrow \gamma'_j$ 
12:    parse  $\text{tx}[\tilde{\gamma}_j] \leftarrow \gamma_j$ 
13:    if  $V(\tilde{l} || \text{tx}[\tilde{\gamma}_j]) = \text{true} \wedge r_i \leq$ 
       $r_j + v$  then:
14:       $\tilde{l} \leftarrow \tilde{l} || \text{tx}[\tilde{\gamma}_j]$ 
15:    end if
16:  end for
17: end for
18: return  $R(\tilde{l})$ 

```

4.7 Security Analysis

Now we state our main theorem and show that the compiled transaction ledger protocol Π' compiled by our CPR-compiler Φ is indeed CPR if the underlying protocol Π is robust.

Theorem 1. *Let Π be a robust transaction ledger protocol. Then, for all expected incentive compatible environments \mathcal{Z} , the compiled transaction ledger protocol $\Pi' \leftarrow \Phi(\Pi)$ achieves CPR.*

The intuition of the proof is to show that dictatorial miners maximize their utility by behaving honestly in all expected incentive compatible environments. If all dictatorial miners behave as honest miners we can conclude that Π' is CPR. The intention is that no single miner nor coalition of miners is able to gain any a-priori knowledge of the transaction content by the time they can decide about including the transactions. Additionally, the construction deincentivizes *delaying* transactions until the miners can learn the content, due to the invalidation of 'too old' transactions. Therefore, dictatorial miners fear of missing out on the transactions and the associated expected profit if they try to learn the content first. The full analysis is deferred to Appendix C.

5 Discussions

In this section we discuss several aspects of our results.

Coercion Resistance in CPR Transaction Ledger Protocols. As shown in our analysis, rational liveness can be achieved if the dictatorial miners gain no a-priori information about the content of candidate transactions. However, miners may try to coerce users in order to make them reveal the content of their transactions a-priori. By doing so, the miners would again be able to enforce their preferences on the transactions. While coercion resistance is to the best of our knowledge a new issue in transaction ledger protocols, it is quite well known in voting protocols [11, 18]. However, the techniques from the voting literature does not seem to apply in our setting. The inability of a user to prove the content of its transaction to a miner would not solve the problem of coercion. Recall that all transactions are eventually opened in the sanitized chain. Thus, a miner could simply establish a contract where a user commits to the content that it discloses before hand, and gets punished if the time-locked transaction opens to something else later on. This coercion attack is possible in this context since in transaction ledger protocol it is very unlikely that another user sends a transaction with the same content. In voting on the other hand it is very much possible that there is another ballot with the same vote. We believe that coercion-resistance in the setting of transaction ledger protocols is an interesting open problem left for future work.

Performance Considerations. In our compiler, every transaction is included as a time-lock puzzle in the control chain, while the solution must be provided later in the sanitized chain, once the puzzle is deep enough in the control chain. This requires miners to solve time-lock puzzles for every transaction. While this additional computational effort burdens the miners, we note that the computational cost *per transaction* is constant (unlike, e.g. PoW). Nevertheless, it is possible to improve the efficiency significantly in practice. For example, one could allow the issuer of the transaction to reveal the puzzle solution once the puzzle is deep enough in the control chain, such that the miners only have to solve the puzzles for non responsive users. Alternatively, recent advancements in time-lock puzzles

yield promising results. In particular, Abadi and Kiayias [1] recently proposed a construction for “chained time-lock puzzles” that allows to compose multiple puzzles into a single one, hence dramatically reducing the computational effort of solving multiple the puzzles.

Invalid and Conflicting Transactions. Naturally, it is hard to decide if a time-lock puzzle contains a transaction that does not contradict any other transaction already in the chain. However, we stress that this is not an issue for the safety and correctness of the ledger since it is still possible to deterministically consider only valid transactions in the sanitized chain. Nevertheless, the inclusion of inconsistent transactions in the control chain might be undesirable in practice. Therefore, flooding the ledger with time-lock puzzles of invalid transactions should be de incentivized. Flooding attacks are usually de incentivized by a fee paid by the sender of a transaction [8, 26]. To this end, in our construction, a time-locked transaction is associated with a ledger account. This associated ledger account can be charged fees for including the time-locked transaction in the control chain, independently of the content hidden inside the time-lock puzzle. Note, that this fee does not necessarily replace any fees for executing the transactions’ contents hidden inside the time-lock puzzle. This assumes that users have access to unlinkable ledger accounts, such that the associate ledger account does not reveal any information about the content hidden inside the time-lock puzzle. This can usually be achieved using anonymization and mixing techniques [6, 29]. However, as in Garay et al. [16], the design of a concrete fee mechanism is outside the scope of this work.

Targeted Censorship. We stress that in this work we are not concerned with censorship targeted at individuals, but we only consider preferences over the contents of the transactions. We note that the full anonymization of transactions requires not only protocol level anonymization but also network level anonymization, what is known to be a hard problem in practice. Dictatorial miners, might be able to gain some information about the transaction content form the network layer of the protocol, for example learning the sender node of a transaction in the underlying network. However, to protect against some level of censorship against individuals, one can still run the CPR protocol over TOR [13].

Alternative Approaches. A different approach to achieve CPR could be by leveraging threshold cryptography or multi-party computation [25]. However, even if threshold cryptography is clearly capable of preventing an unqualified set of miners from taking rational actions, it also inherently defines a coalition of miners that can. As pointed out by Daian *et al.* [10], any coalition that yields higher utility should be expected to be formed eventually. Hence, we rely on time-lock puzzles which are inherently coalition resistant by design. Choosing time-lock parameters in a way that no miner can learn the content a-priori while on the same hand guaranteeing that every miner can be expected to provide the solutions is practically challenging. In particular, due to hardware differences,

some miners may perform the required sequential computation faster than others. Also, aligning the delay parameter of the time-lock puzzle with the desired block creation time of the underlying transaction ledger protocol might be challenging, especially for probabilistic block creation times [8, 26]. Therefore, a practical instantiation should reflect this by considering a gap in which miners are expected to provide the solutions for time-lock puzzles. Another approach to the computational time-lock puzzle is proposed by Liu *et al.* [23]. They propose a construction of a “time-release encryption” relative to a reference clock using witness encryption. With their construction it is possible to encrypt a transaction such that its plaintext is released, e.g., at a predefined blockdepth of the ledger. While this solves the challenge of choosing time-lock puzzle parameters, the implementation of such scheme would be rather impractical, since the size of the witness used for decryption is approximately the size of the entire blockchain.

6 Conclusion

In this work we investigate the setting where “dictatorial miners” can use their a-priori knowledge of transactions’ content to alter the set and order of candidate transactions in their most favorable way to improve their utility. To this end, we introduce the model of dictatorial miners that may deviate from honest behavior by suppressing or reordering transactions selectively depending on their content. We incorporate dictatorial miners in the transaction ledger protocol modeled by Garay *et al.* [16] by replacing honest miners with dictatorial ones. In that vein, we show that a transaction ledger protocol can achieve content preference robustness by guaranteeing rational liveness and rational transaction order preservation. We show that this can be achieved if dictatorial miners cannot learn the contents of transactions before they are in the common-prefix of the chain. In particular, we provide a construction for a CPR compiler that can transform any generic robust transaction ledger protocol into a CPR ledger protocol by leveraging time-lock puzzles.

Acknowledgements. We thank the anonymous reviewers for their effort and valuable comments that helped to improve this work. This work was partly supported by the German Federal Ministry of Education and Research (BMBF) iBlockchain project (grant nr. 16KIS0902) and by the German Research Foundation (DFG) via the DFG CRC 1119 CROSSING (project S7).

A Related Work

In this section we discuss some related works and compare them with our results.

Bitcoin Incentive Compatibility. Badertscher *et al.* [2] showed that Bitcoin satisfies the properties of persistence and liveness (as defined in [16]) in presence of a rational majority. However, in their work the utilities of the rational participants are restricted to a natural class of incentives for the miners, such as fees

and block rewards, explicitly excluding preferences over transaction contents. In this work we extend their results and explicitly focus on utilities based on transaction contents.

Order-Fairness for Byzantine Consensus. Kelkar *et al.* [19] deal with the issue of order fairness in transaction ledger protocols. They point out that consistency and liveness do not protect against malicious manipulation of the received order of transactions. This implies that the resulting ordering of transactions does not necessarily reflect the received ordering. Their proposed solution provides block order fairness, which says that if sufficiently many miners receive a transaction tx before tx' then no honest miner can report tx' in a block before tx . Further, they show that it is not possible to guarantee received order fairness, consistency and liveness at the same time.

Moreover, [19] gives a positive result for a slightly weaker definition of order fairness under the strict assumption that a fraction of the parties behave honestly, i.e., the honest parties will not alter the order of transactions under any circumstances. In comparison, our model does allow every miner to alter the order of candidate transactions, or even suppress them, for the sake of individual profit. Our construction ensures that miners are indifferent between transactions they are supposed to include into the ledger, and do therefore not expect a higher utility for altering the transaction's order. Note however that this does not contradict the impossibility result of Kelkar *et al.* [19].

Censorship Resistant Consensus. Miner's suppression of transactions was already addressed by Miller *et al.* [25]. In order to achieve censorship resilience they propose a BFT consensus protocol combined with threshold encryption. In the proposed construction miners select transactions from their local buffer and encrypt them under the common public key of the threshold encryption. Before decryption the miners exchange and agree on the encrypted transactions. As in [19] the security of the construction is also based on the assumption of an honest fraction of miners. We note that in our model, any qualified set of miners can decrypt the threshold encrypted messages at any time and therefore can learn the plaintext collaboratively for the sake of common and individual profit.

Time-lock Puzzle in Blockchains. Khalil *et al.* [20] provide an implementation of a trustless centralized exchange based on an underlying blockchain that prevents front-running attacks of the centralized operator and the miners of the blockchain by using time-lock puzzle. The idea of their construction is that the set and the ordering of bids and offers is determined before the plaintexts are revealed.

Deuber *et al.* [12] use time-lock puzzles to ensure opening of commitments. In [12] they propose a minting mechanism based on waiting time auctions. In order to ensure that the block creators actually include all the openings for the commitments of bids in a block, it is required that all openings to the commitments are encapsulated in a time-lock puzzle and sent together with the bid transaction. In both works time-lock puzzles are used to ensure that commitments can be opened even if the opening messages get "lost" in the way, e.g. by

a corrupted miner. While both of the previous works utilize time-lock puzzles to ensure openings in their respective ledger application we leverage time-lock puzzles to strengthen the ledger itself by improving its liveness guarantees.

Further, Doweck and Eyal [14] propose a construction for a multi-party timed commitment. In their construction a set of N users engage in an interactive commitment protocol with a single coordinator to commit to a list of messages by the users that can be revealed by the coordinator at a later time. Their construction is based on El-gamal encryption with a randomly sampled public key of a small group size, where the private key is revealed by the coordinator by brute force. Additionally, they provide a construction for a transaction ledger protocol that leverages their multi-party timed commitment. However, their construction requires the users to engage in interactive commitment protocols with one or even several miners leading to a significant communication overhead especially for higher numbers of users. Moreover, the searching for an El-gamal private key can be parallelized, offering no lower bound of operations under miners' coalition. Our construction on the other hand let users publish and propagate their transactions as commonly done in transaction ledger protocols.

Bribery and MEV Attacks. There are various incentive based attacks utilizing rational miners that intend to either revise, reorder or to exclude certain transactions from the ledger. On a high level, all these attacks are dynamics that might influence a rational miner's preference over transaction content. In [22], Liao and Katz show an attacker that incentivizes forking the main chain using high transaction fees. Moreover, McCorry *et al.* [24] present a different bribery contract that makes the miners change their mining strategy. In their work, miners' utility depends on the attackers bribe rather than on hidden content preferences. For example, the goldfinger attack of [24] incentivizes miners to mine empty blocks independent of the content of any available transaction.¹³

Bribery attacks that incentivize miners to suppress transactions based on their content are proposed by Winzer *et al.* [31] and Tsabary *et al.* [30]. These types of attacks can be prevented if the dictatorial miners are *not able* to choose transactions based on their content.

Daian *et al.* [10] introduced *Time-bandit attacks* where adversarial miners can fork the blockchain by utilizing MEV opportunities. The attack works by leaving MEV opportunities in the main chain for other miners to claim, thus incentivizing other rational miners to fork the chain to claim the MEV opportunity. Similarly to [22], this subsidizes a 51% attack. Miners may be incentivized to break consensus if the block rewards are not enough in comparison to the MEV [10] opportunities. While our construction does not entirely prevent this type of attack, it mitigates it by making the attack more costly for the miner to pull off; the required fork would to claim the MEV would be considerably deeper, making it less profitable.

¹³ Clearly, this attack can not be prevented by our construction and outlines the limits we will elaborate in this work; dictatorial miners might suppress transactions independent of the content if they expect to improve their utility by doing so.

Sandwich attacks are a common predatory trading strategy in which a miner or a trader “wraps” a victim’s transaction between two adversarial transactions [10]. If the market price of an asset is expected to rise/fall after the execution of a large pending transaction, the adversary may extract value by inserting its own transaction right before/after the spotted pending transaction. Our construction prevents sandwich attacks since the attacker is not able to spot a target transaction and sandwich it at the same time.

Finally, Judmayer et al. [17] showed that it is almost impossible to determine the exact globally available MEV opportunities at a certain point in time, and that a narrow definition of MEV fails to capture all extractable value occasions of other actors, the emerging network dynamics, or the probabilistic nature of permissionless cryptocurrencies. In that vein, we consider the complexity of MEV by assuming the exact utility of miners for including transaction to be unknown.

B Transaction Ledger Protocol

According to Garay *et al.* [16] a transaction ledger aims at keeping a record of monetary accounts and its associated balance; a transaction record in the ledger is typically (but not limited to) an instruction to move balances between accounts. A transaction ledger is represented as a vector of blocks $l = (\mathcal{B}_1, \dots, \mathcal{B}_d)$, where each block $\mathcal{B}_i = (tx_1, \dots, tx_n)$ is a vector of transactions $tx \in \mathcal{T}$. \mathcal{T} denotes the set of valid transactions. Appending a transaction tx to a vector l is denoted by $l||tx$. Also, appending a vector of transactions \mathcal{B} to another vector l is denoted as $l||\mathcal{B}$. $tx_{i,j}$ denotes transaction tx_j in block \mathcal{B}_i . As a ledger is a vector of transactions, we simply denote it as $l = (tx_1, \dots, tx_m)$ omitting the block numbers when clear from the context.

The transaction ledger protocol is executed by a set of miners \mathcal{M} in the presence of a PPT adversary \mathcal{S} , and driven by a PPT environment \mathcal{Z} . The protocol execution takes place in rounds. The environment provides inputs to all parties and receives outputs, while the attacker might fully corrupt some of the miners. Each honest miner M_i maintains its own local copy of the chain l_i . Further, an honest miner M_i process a local buffer $X_i := (tx_1, \dots, tx_e)$, that are candidate transactions to be incorporated into the ledger l_i provided by the environment \mathcal{Z} . In [16], a transaction ledger protocol is defined by the transaction generation oracle TxGen, the set of valid ledgers \mathcal{L} and by the three interface functions $V(\cdot), I(\cdot), R(\cdot)$.

The transaction generation oracle TxGen generates transactions on behalf of the users \mathcal{P} which are abstracted by the environment \mathcal{Z} . It is defined with respect to the set of valid transactions \mathcal{T} , the set of valid contents Γ , (which denotes the set of content information with semantic value for the ledger, e.g., “account A increases its balance by 10 monetary units”) and the set of ledger accounts \mathcal{A} . Note that a user P_i might be associated with multiple ledger accounts. During the execution of the transaction ledger protocol, TxGen can be accessed by the environment \mathcal{Z} and it generates transactions that are provided to the miners and the adversary \mathcal{S} . Upon receiving a message $(\text{IssueTx}, \gamma, P)$ from the environment \mathcal{Z} , TxGen generates a unique transaction $tx[\gamma] \in \mathcal{T}$, where $tx[\gamma]$ denotes a

transaction tx that contains an encoding of content γ . After that, TxGen sends $(\text{Issued}, tx[\gamma], A)$ for some ledger account $A \in \mathcal{A}$ to every miner and \mathcal{S} .

On the other hand, the three interface functions $V(\cdot), I(\cdot), R(\cdot)$ are defined as follows:

- $V(l)$: The content validation predicate, upon input a sequences of transactions $(tx_1[\gamma_1], \dots, tx_m[\gamma_m])$ checks whether all the transactions constitute a semantically valid ledger. Formally, $V(\cdot)$ defines the set of valid ledgers \mathcal{L} and checks if $l \in \mathcal{L}$, e.g. $V(l)$ checks if there are no conflicting transactions in l .
- $R(l)$: The chain reading function returns a semantic interpretation of the contents $(\gamma_1, \dots, \gamma_n)$, e.g. a list of account addresses and balances Upon receiving a ledger $l = (tx_1[\gamma_1], \dots, tx_n[\gamma_n])$, and if $V(l) = 1$.
- $I(l, \mathbf{X}, r)$: Upon receiving a ledger and a buffer of local transactions in some round r the input contribution function creates some new block $\mathcal{B} = (tx_1[\gamma_1], \dots, tx_e[\gamma_e])$, where $tx_i \in \mathbf{X}$ and returns $l' := l \parallel \mathcal{B}$.

Moreover, a transaction ledger protocol is called *robust* if the following properties are satisfied:

- *Persistence*: If at any round r an honest miner M_i maintains a ledger that contains a transaction $tx \in \mathcal{T}$ in a block more than $k \in \mathbb{N}$ blocks deep in the chain, then tx occurs at the same position in the chain of all the other honest miners.
- *Liveness*: If a transaction $tx \in \mathcal{T}$ issued by TxGen is input for all honest miners in \mathcal{M} for at least v consecutive rounds, then all honest miners will report this transaction at least k blocks deep into the ledger, for some $k, v \in \mathbb{N}$.

According to [16] a robust transaction ledger protocol can be build on top of a blockchain backbone protocol that satisfies the properties *common prefix*, *chain quality* and *chain growth* by defining the interfaces I, V, R, TxGen , and \mathcal{L} . In our work we assume the existence of a robust transactions ledger protocol $\Pi = (I, V, R, \text{TxGen}, \mathcal{L})$ for some liveness parameter v . Therefore, we can waive details of the underlying backbone protocol that is used to implement Π . For more details on the ledger backbone protocol we refer the reader to the paper of Garay *et al.* [16].

C Analysis of Theorem 1

Let $\Pi = (I, V, R, \text{TxGen}, \mathcal{L})$ be a robust transaction ledger protocol executed by a set of miners \mathcal{M} in the presence of a PPT adversary \mathcal{S} driven by some environment \mathcal{Z} , and let $\Pi' = (I', V', R', \mathcal{F}_{\text{tl-TxGen}}, \mathcal{L}')$ be the compiled transaction ledger protocol $\Pi' \leftarrow \Phi(\Pi)$ executed by a set of dictatorial miners \mathcal{M}' in the presence of a PPT adversary \mathcal{S} driven by some environment \mathcal{Z}' . Let $\delta = v$ be the delay parameter of $\mathcal{F}_{\text{tl-TxGen}}$.

At any round r a dictatorial miner M_i receives a transaction buffer \mathbf{X}_i from the environment \mathcal{Z}' and provides an altered transaction buffer \mathbf{X}'_i to the input contribution function $I'(\cdot)$.

In order to show that Π' achieves CPR it is necessary to show that Π' achieves *rational liveness* and *rational transaction ordering*. To this end, we show that a dictatorial miner M_i can not improve its expected utility u_i by deviating from honest behavior. A dictatorial miner M_i behaves honest if $X'_i = X_i$ at any round r . If it is in the best interest of dictatorial miners to behave honest it can be concluded that Π' achieves rational liveness if Π achieves liveness.

Let therefore $X_i = (tx'_1, \dots, tx'_n)$ with $tx'_j = (\text{txid}_j, \tilde{tx}_j, A_j)$ for all $j \in [n]$ be a transaction buffer that is provided to some dictatorial miner M_i in some round r_y for some current ledger $l'_{r_y} \in \mathcal{L}'$ by the environment \mathcal{Z} . Since, \mathcal{Z} is expected incentive compatible it holds that $u_i(l', tx'_j) > 0$ for every $tx'_j \in X_i$. Therefore, it follows that M_i prefers to *include* tx'_j over suppressing it, if $\gamma'_j = (\text{sid}_j, \gamma_j, r_j, P_j)$ associated with tx'_j is sampled by \mathcal{Z} from some common prior distribution over Γ and M_i did not gain any additional information about γ_j . Therefore, a dictatorial miner that is able to reduce its uncertainty about γ_j over the course of some rounds might actually be able to improve its expected utility by *suppressing* tx'_j . Consequently, it would be in a dictatorial miners best interest to learn the content of transactions instead of relying on the common prior expectation.

Since \mathcal{Z} provides transactions tx'_j issued using the functionality $\mathcal{F}_{\text{tl-TxGen}}$ any dictatorial miner is able to learn the content γ'_j associated with tx'_j after at least v rounds after it was issued.¹⁴ However, $\mathcal{F}_{\text{tl-TxGen}}$ does not allow any single miner M_i nor an adversary \mathcal{S} that corrupts any subset of miners to learn γ'_j before v rounds. In particular this means that no single miner nor any coalition of miners is able to reduce its uncertainty about γ'_j before v rounds. However, a dictatorial miner M_i could still improve its expected utility by *delaying* every transaction tx'_j it receives in some round r_y for v rounds so it can learn its contents. To this end, the chain reading function $R'(\cdot)$ checks for every transaction $tx'_j = (\text{txid}_j, \tilde{tx}_j, A_j)$ with associated content $\gamma'_j = (\text{sid}_j, \gamma_j, r_j, P_j)$ included in some block \mathcal{B}^{r_y} created in round r_y if $r_y \leq r_j + v$. If not, γ'_j is ignored by $R'(\cdot)$. Consequently, whenever a dictatorial miner M_i receives a transaction tx'_j in some round r_y for the first time and decides to *delay* this transaction for at least 1 round, it knows that γ'_j associated tx'_j will be ignored by $R'(\cdot)$. Since any content γ'_j that is ignored by $R'(\cdot)$ is treated as if the corresponding transaction tx'_j was not included at all *delaying* a transaction yields the same expected utility as *suppressing* it for every dictatorial miner M_i , every transaction tx'_j and every ledger $l'_{r_y} \in \mathcal{L}'$. Since, \mathcal{Z} is expected incentive compatible it can be concluded that every dictatorial miner M_i prefers to *include* any transaction tx'_j in the round it received it first. Therefore, any miner M_i will include any transaction $tx'_j \in X_i$ into X'_i in any round. Therefore, Π' executed by a set of dictatorial miners \mathcal{M} in presence of an adversary \mathcal{S} driven by an expected incentive compatible environment \mathcal{Z} achieves *rational liveness*. Moreover, let $X_i^{r_y}$ be the set of transactions in X_i that miner M_i received for the first time in that round r_y . Since for every transactions $tx'_j \in X_i^{r_y}$ the transaction tag \tilde{tx}_i and the associated account A are chosen uniformly at random and do not reveal any information

¹⁴ Note that, since \mathcal{Z} is expected to provide the inputs to the dictatorial miners the miner M_i is expected to receive any transaction tx'_j in the same round it is issued.

about the associated content γ'_i any dictatorial miner M_i must be indifferent between either including some transaction tx'_1 in some ledger $l'_{round_y} || tx'_0$ or including some transaction tx'_0 in some $l'_{round_y} || tx'_1$ for every pair of transactions $(tx'_0, tx'_1) \in X_i^{r_y}$ and every ledger l'_{round_y} . Therefore, it can be concluded that Π' also achieves *rational transaction preservation*.

References

1. Abadi, A., Kiayias, A.: Multi-instance publicly verifiable time-lock puzzle and its applications (2021)
2. Badertscher, C., Garay, J., Maurer, U., Tschudi, D., Zikas, V.: But why does it work? A rational protocol design treatment of bitcoin. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 34–65. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_2
3. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: a composable treatment. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 324–356. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_11
4. Bano, S., et al.: SoK: consensus in the age of blockchains. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, pp. 183–198 (2019)
5. Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: TARDIS: a foundation of time-lock puzzles in UC. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12698, pp. 429–459. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77883-5_15
6. Ben-Sasson, E., et al.: Zerocash: decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 459–474. IEEE Computer Society Press (2014). <https://doi.org/10.1109/SP.2014.36>
7. Boneau, J.: Why buy when you can rent? In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D., Brenner, M., Rohloff, K. (eds.) FC 2016. LNCS, vol. 9604, pp. 19–26. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_2
8. Buterin, V.: Ethereum: a next-generation smart contract and decentralized application platform (2014). <https://github.com/ethereum/wiki/wiki/White-Paper>
9. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (2001). <https://doi.org/10.1109/SFCS.2001.959888>
10. Daian, P., et al.: Flash boys 2.0: frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy (SP), pp. 910–927. IEEE (2020)
11. Delaune, S., Kremer, S., Ryan, M.: Coercion-resistance and receipt-freeness in electronic voting. In: 19th IEEE Computer Security Foundations Workshop (CSFW2006), p. 12. IEEE (2006)
12. Deuber, D., Döttling, N., Magri, B., Malavolta, G., Thyagarajan, S.A.K.: Minting mechanism for proof of stake blockchains. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 2020. LNCS, vol. 12146, pp. 315–334. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57808-4_16
13. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The Second-generation Onion Router. Tech. rep, Naval Research Lab Washington DC (2004)

14. Doweck, Y., Eyal, I.: Multi-party timed commitments. arXiv preprint [arXiv:2005.04883](https://arxiv.org/abs/2005.04883) (2020)
15. Eskandari, S., Moosavi, S., Clark, J.: SoK: transparent dishonesty: front-running attacks on blockchain. In: Bracciali, A., Clark, J., Pintore, F., Rønne, P.B., Sala, M. (eds.) FC 2019. LNCS, vol. 11599, pp. 170–189. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43725-1_13
16. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
17. Judmayer, A., Stifter, N., Schindler, P., Weippl, E.: Estimating (miner) extractable value is hard, let's go shopping! Cryptology ePrint Archive, Report 2021/1231 (2021). <https://ia.cr/2021/1231>
18. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Chaum, D., et al. (eds.) Towards Trustworthy Elections. LNCS, vol. 6000, pp. 37–63. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12980-3_2
19. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for Byzantine consensus. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 451–480. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_16
20. Khalil, R., Gervais, A., Felley, G.: TEX - A securely scalable trustless exchange. Cryptology ePrint Archive, Report 2019/265 (2019). <https://eprint.iacr.org/2019/265>
21. Kroll, J.A., Davey, I.C., Felten, E.W.: The economics of bitcoin mining, or bitcoin in the presence of adversaries. In: Proceedings of WEIS, vol. 2013, p. 11 (2013)
22. Liao, K., Katz, J.: Incentivizing blockchain forks via whale transactions. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 264–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_17
23. Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Des. Codes Crypt. **86**(11), 2549–2586 (2018). <https://doi.org/10.1007/s10623-018-0461-x>
24. McCorry, P., Hicks, A., Meiklejohn, S.: Smart contracts for bribing miners. In: Zohar, A., et al. (eds.) FC 2018. LNCS, vol. 10958, pp. 3–18. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-662-58820-8_1
25. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of BFT protocols. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 31–42. ACM Press (2016). <https://doi.org/10.1145/2976749.2978399>
26. Nakamoto, S.: Bitcoin: A Peer-to-peer Electronic Cash System. Tech. rep, Manubot (2019)
27. Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: how dark is the forest? arXiv preprint [arXiv:2101.05511](https://arxiv.org/abs/2101.05511) (2021)
28. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
29. Ruffing, T., Moreno-Sanchez, P., Kate, A.: P2P mixing and unlinkable bitcoin transactions. In: NDSS 2017. The Internet Society (Feb/Mar 2017)
30. Tsabary, I., Yechieli, M., Eyal, I.: MAD-HTLC: Because HTLC is crazy-cheap to attack. arXiv preprint [arXiv:2006.12031](https://arxiv.org/abs/2006.12031) (2020)
31. Winzer, F., Herd, B., Faust, S.: Temporary censorship attacks in the presence of rational miners. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 357–366. IEEE (2019)



An Empirical Analysis of Security and Privacy Risks in Android Cryptocurrency Wallet Apps

I. Wayan Budi Sentana^(✉) , Muhammad Ikram^(✉) ,
and Mohamed Ali Kaafar^(✉) 

Department of Computing, Macquarie University, 4 Research Park Drive, Sydney,
NSW 2113, Australia

i-wayan-budi.sentana@hdr.mq.edu.au

Abstract. A cryptocurrency wallet app is a piece of software that manages, stores, and generates private keys of cryptocurrency accounts. With the provision of services such as easy access to transaction history, and checking account balance besides transmissions of new transactions in distributed networks such as Blockchains, cryptocurrency wallet apps gain unprecedented popularity which in turn attracts malicious actors to attack users resulting in loss of cryptocurrency assets and leakage of sensitive user data. This paper presents the first large-scale study of Android cryptocurrency wallet apps. We surveyed apps on Google Play to detect and extract meta-data and application packages of 457 cryptocurrency wallet apps. We perform several passive and active measurements designed to investigate the security and privacy features to study the behavior of cryptocurrency wallet apps. Our analysis includes investigating cryptocurrency wallet apps' third-party embedding, malware presences, and exfiltration of users' sensitive data to third-parties. Our study reveals vulnerabilities and privacy issues in cryptocurrency apps including the insecure use of HTTP to serve transactions.

Keywords: Cryptocurrency Wallet · Static Analysis · Dynamic Analysis · User-review Analysis

1 Introduction

Cryptocurrency wallet applications (or wallet apps) for mobile devices are used to securely store cryptocurrency assets and enable users to control their assets over private keys. Like using any type of mobile application, users of wallet apps are possibly faced with security and privacy vulnerabilities rendered by developers' coding practices and privacy practices related to their business models. However, as wallet apps are involved in controlling users' financial assets, the vulnerabilities become more severe if they are leveraged by attackers. For instance, adversaries could leverage the embedded third-party libraries and request permissions of the app to steal important information such as private keys. Hence,

it is important to assess wallet apps' behaviors for potential security and privacy issues to inform users as well as developers. Recently, several works have studied security and privacy issues of wallet apps [16, 19, 23, 33]. Nevertheless, their analysis focuses on some characteristics such as apps requested permission, and privacy leakage, and are limited to a few existing apps.

To provide a comprehensive assessment of the security and privacy issues of 457 Android wallet apps, we perform the first large-scale study of the privacy and security features of wallet apps on Google Play. We first collect wallet apps available in the Google Play store. We use static analysis and dynamic analysis to investigate the privacy and security features of these apps. In static analysis, we evaluate the source code of each app to identify the requested permissions, third-party libraries embedding, malware presence, and anti-analysis adoption. Whereas in dynamic analysis, we monitor the application during execution and capture the network traffic to observe the application interaction. Moreover, we analyse wallet apps' privacy policies for compliance and evaluate users' perceptions by investigating users' comments. To foster future research, we release our code and dataset used in this paper to the research community: <https://walletapps2021.github.io/>.

The main contributions of our work are as follows:

Source Code Analysis. We collect a corpus of 457 wallet apps from Google Play and illuminate their evolution and popularity on Google Play (Sect. 2.2). We systematically analyse wallet apps' source codes and find potential security issues spanning from requesting dangerous permission, embedding third-party libraries for advertising and tracking purposes, presence of malware code, and using anti-analysis techniques (Sect. 4).

Our analysis identified 8 (1.7%) apps using sensitive permission such as `android.permission.DOWNLOAD_WITHOUT_NOTIFICATION` in their code which, once requested, enables the app to download any file or malware executables without user consent, 25 (5.4%) wallet apps embed malware code in their source code according to VirusTotal [43]. We also found 13 (2.8%) apps embed Cross-Library Data Harvesting (XLDH) library [44] which "illegally" extracts user information from legitimate libraries such as Facebook, Google, Twitter, and Dropbox.

Apps' Network Traffic Analysis. We investigate the runtime and network behavior of the wallet apps by installing them into an Android phone and navigating the app's activities while running on the phone (Sect. 5). Our analysis detected broad evidence of potential security and privacy issues of the apps, including data leakages, using unencrypted transmission via HTTP, and requesting third-party domains containing advertisers and trackers. For instance, we detected 148 apps transmitting their traffic via unencrypted HTTP protocol. Moreover, we identified 15 apps (e.g., `com.btcc.mobiwalletand` and `com.coinburp.mobile`) sharing user credentials and device information with third-parties via HTTP.

Privacy Policy and Users’ Reviews Analysis. We analyse potential compliance issues of apps by evaluating apps’ privacy practices and investigating app user reviews (Sect. 6). Particularly, our analysis detected 87 (19%) apps failed to provide privacy policy links for their user and 78 (17%) violated the privacy policy related to sharing users’ information with third-parties.

2 Background and Data Collection Methodology

2.1 Background

Cryptocurrency is a digital asset that uses cryptography to ensure its creation security and transaction security [45]. There are over 2,500 different kinds of cryptocurrencies now, where the most well-known cryptocurrencies are “Bitcoin” and “Ethereum” which are traded at numerous cryptocurrency exchanges or marketplaces. Cryptocurrency exchanges offer trade services among cryptocurrencies. There are centralised exchanges (governed by a company), decentralized exchanges (provided an automated process for peer-to-peer trades), and hybrid exchanges [45].

The traded cryptocurrencies are normally kept at *wallet* in the form of hash values termed as *wallet addresses* [23]. Each address is corresponding to a pair of keys: public and private. The public key is used for external transactions such as sending or receiving cryptocurrencies. In order to prove the ownership of the cryptocurrency, each transaction is signed with a private key. If a user loses their private keys, they will lose their associated cryptocurrency assets. Anyone who gains the private key of a public address can authorize a transaction.

2.2 Cryptocurrency Wallet Apps Collection on Google Play

Google Play neither lists wallet applications as distinct apps’ categories nor its search functionality yields all wallet applications. To collect and identify all possible wallet apps, we develop a crawling methodology. First, we query Google Play search with cryptocurrency-related keywords to collect a seed cryptocurrency wallet app. The keywords consist of “crypto”, “cryptocurrency”, “bitcoin”, “coin”, “Ethereum”, “wallet”, and others coin abbreviation such as “BTC”, “ETH” and “DOGE“. We then use Google Play Store Scraper API¹, a tool for scraping and parsing application data from the Google Play Store², to recursively crawl *similar* apps of the seed app IDs. By using this method, we found 3,629 app IDs. The API also returned the apps’ meta-data such as the app’s title, average rating, user reviews, descriptions, and categories to be further used for the app’s selection. We then manually checked each app’s description and filtered only cryptocurrency wallet apps. Finally, we obtain 457 free cryptocurrency wallet apps. We leverage `gplaycli` [25] to download the application packages (APKs) from Google Play of 457 cryptocurrency wallet apps. Given that Google Play

¹ <https://pypi.org/project/google-play-scraper/>.

² <https://play.google.com/store>.

does not report the actual release date of the apps but their last update, we use the date of their first comment as a proxy for their release date. For 63 (13.8%) apps without any user reviews as of this writing, we determine the approximate release date by their last update.

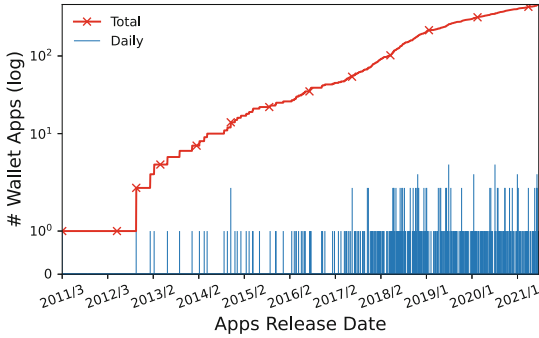


Fig. 1. Evolution of the analyzed cryptocurrency wallet apps on Google Play.

Figure 1 shows that cryptocurrency wallet apps have increased four-fold since February 2018. Figure 2 depicts the distributions of the number of installs and an average rating of the analyzed cryptocurrency wallet applications on Google Play. We found that 21.8% (100) of the applications have at least 50,000 installs. We also observe that 51.3% (234) of the applications have at least 3.0 average ratings showing that the vast majority of the applications are positively rated by users. Conversely, we noted that 63 (13.8%) apps without any reviews have an average rating of 0.0 and have at most 500 installs.

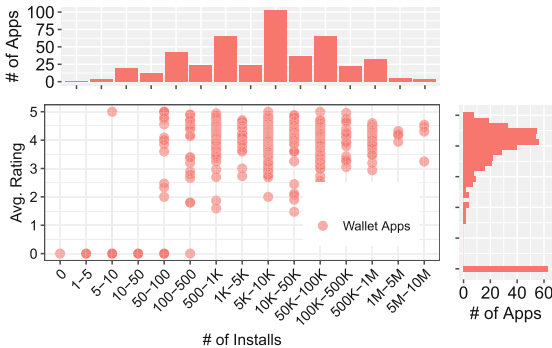


Fig. 2. Distribution of the number of installs and average app's rating of the analyzed cryptocurrency wallet apps on Google Play.

3 Assessment Methodology

We use a set of custom-built tools to assess source codes and network runtime behaviors of the apps for potential security and privacy issues. The following sections describe our analysis categorised into static, dynamic, and compliance analysis.

3.1 Static Analysis

We assess wallet apps for requesting sensitive permission, presence of third-party tracking libraries and malware codes, use of security certificates, anti-analysis methods, compliance and user reviews analysis, and leakage of sensitive data to third-parties.

Permission Analysis. To get the resources it needs during runtime, each wallet app must declare all of its resource requirements in the `AndroidManifest.xml` file. We use `apktool` to decompress the APKs of the analysed apps and extract all permissions declared in `<uses-permission>` and `<service>`³ tags. We analyse the requested permissions for potential vulnerabilities and wallet apps' capabilities for possible exploits.

Tracking Libraries. By inspecting the decompressed source code of each app, we examine the presence of embedded third-party libraries. To identify which libraries are associated with tracking, analytics, and advertising services, we aggregate the manually curated list of 383 tracking and advertising libraries from [15, 20, 21, 40]. In particular, we inspect the APK file of an app to determine sub-directories and match them with our list of tracking and advertising libraries. If there is a match between the sub-directory and an entry of the third-party libraries, the app is deemed to be used by the corresponding third-party library. Given that apps may use obfuscation methods to hide the names of third-party libraries [8, 14], we consider our results as a lower bound on the presence of third-party libraries in cryptocurrency wallet apps.

We also analyzed the existence of *suspicious* Cross-Library Data Harvesting (XLDH) libraries in cryptocurrency wallet apps. XLDH is a type of library that steals user information from legitimate libraries such as Facebook, Google, Twitter, and Dropbox. XLDH libraries actively scan the legit libraries' existence and extract important information from the library's Software Development Kit (SDK) embedded in the same apps. Extracted information including user access token, user name, advertisement ID, and user image is then sent to the XLDH libraries server. Due to the illegal operation, Facebook has taken legal action against several companies providing this XLDH library [44]. To identify the emergence of XLDH libraries, we scanned the third-party libraries embedded by the cryptocurrency wallet app and compared the results against a list published by [44].

³ Note that service permissions are requested at runtime to enable specific functions such as connecting to a VPN network.

Malware Presence. We leverage VirusTotal [43], an online solution that aggregates the scanning capabilities provided by over 70 antivirus engines (AV), to detect malware activities in the apps. We automate our malware analysis by using VirusTotal’s Report API to retrieve the malware detection results. After completing the scanning process for a given app, VirusTotal generates a *positive* report that indicates which of the participating AV scanning tools detected any malware activity in the app and the corresponding malware signature (if any).

Out of 457 apps, 153 apps have a size more than the VirusTotal API threshold (32 MB) which cannot directly scan for by using the API. Fortunately, there are 35 apps that have been analyzed by the VirusTotal previously. Hence, we manually upload 118 apps to the VirusTotal website to scan for malware presence in the apps.

Anti-analysis Detection. We determine whether an app uses any means of evading, obscuring, or disrupting the analysis of parties other than the application developers. In fact, malware developers rely on these techniques to evade basic analysis layers of application market stores such as Google Play [6]. For example, He et al. [17] found that 52% of their malware samples leveraged anti-analysis techniques to evade, obscure, or disrupt analysis methods. We use APKiD [32] tool to obtain a list of anti-analysis techniques such as “manipulator”, “anti-virtual machine”, “anti-debug”, “anti-disassembly”, “obfuscator”, and “packer” (cf. Appendix A for further details).

3.2 Network Measurements

To investigate the runtime and network behavior of the cryptocurrency wallet apps, we manually install each app on a Huawei GR5 phone running Android Version 7.0 and navigate the app running on the phone. To capture the network traffic generated by each app, we run *mitmproxy* [26] at a WiFi access point to intercept all the traffic being transmitted between the mobile phone and the Internet. For each of the analysed apps, we manually navigate the app *activities* such as login and clicking on buttons to potentially execute app components. Our manual tests last for at least three minutes per app. We inspect the captured network traffic of each app for data leakages, communication with third-party domains, and HTTPS adoption.

3.3 Compliance and User Comments Analysis

Privacy Policy Analysis. We aim to examine the compliance of the apps’ developers in two categories: (i) providing a privacy policy link, and (ii) adhering to the privacy policy in terms of sharing user information with third-parties (TP). In the first category, we mark an app as violating the privacy policy agreement if it does not provide a privacy policy link or does provide a misleading link such as a broken link or an inaccessible link. In the second category, we mark an app failing to adhere to the privacy policy if it declares not to share user

information with the TPs in the privacy policy descriptions, but in fact, sharing them during the app’s execution. We develop a tool to retrieve and extract the privacy policy description of each app using the `BeautifulSoup` Python library. We then use a machine learning classification model [46] to classify if the app claims to collect and share user information with TPs in its privacy policy.

User Reviews Analysis. Unlike most financial applications such as banking apps developed and released by official financial institutions such as banks [3], cryptocurrency wallet apps can be developed and released by any party or developer. Another key feature of the wallet apps is that there is no underlying asset like the banking system. Users choose cryptocurrency wallets only based on trust. Thus, in this study, we make user review one of the important parameters in analyzing security issues and the possibility of privacy leaks from users of cryptocurrency wallet apps. We create scripts to aggregate user reviews (comments) for each app and classify them into positive (4- and 5-star), neutral (3-star), and negative (1- and 2-star) reviews according to the number of stars that the user chose for the app.

Table 1. Top 10 dangerous permissions requested by analyzed cryptocurrency wallet apps in our dataset.

No	Permission Name	# of Request (%)
1	CAMERA	399 (87.3%)
2	WRITE_EXTERNAL_STORAGE	334 (73.1%)
3	READ_EXTERNAL_STORAGE	268 (58.6%)
4	INSTALL_PACKAGES	146 (31.9%)
5	READ_PHONE_STATE	97 (21.2%)
6	ACCESS_COARSE_LOCATION	96 (21.0%)
7	ACCESS_FINE_LOCATION	96 (21.0%)
8	RECORD_AUDIO	82 (17.9%)
9	READ_CONTACTS	68 (14.9%)
10	GET_ACCOUNTS	54 (11.8%)

4 Static Analysis Result

4.1 Permission Analysis

Android classifies permission requests into three categories including *normal* permissions, *dangerous* permissions, and *signature* permissions [10]. We found 8,339 permission requests from 457 wallet apps. We group the permission requests according to the resource sensitivity access levels. Additionally, Android also allows third-parties to develop their own permissions with names and syntax tailored to the standard developer library. We found that 1,755, 2,802, and 570

permission requests fall into the dangerous category, normal category, and signature permissions respectively. In addition, we found that 3,212 permission requests were not covered by the permission level list listed in [10]. We then group these permission requests into the customized/third-party permission category for our further analysis.

Table 1 shows the 10 most dangerous permissions requested by cryptocurrency wallet apps. We detected that 399 (87%) apps request permission to access the device’s camera to activate the camera during the user document verification process including taking a photo of a legitimate government-issued ID and scanning the wallet address in the form of a QR code. We also found 82 (18%) apps requesting voice recordings as shreds of evidence that users give consents for them to store user data.

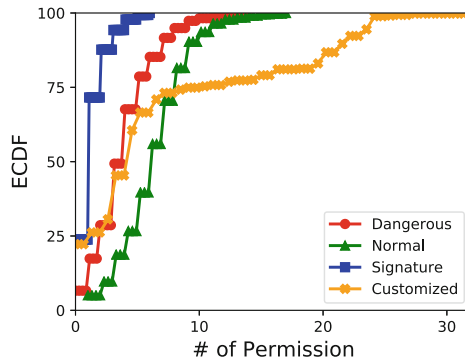


Fig. 3. Empirical cumulative distribution function (ECDF) of permission per app across various categories as per Android Official API [9].

On average, each app requests 18 permissions, which consist of 4 dangerous permissions, 7 customized permissions, 6 normal permissions, and 1 signature permission. Specifically, Fig. 3 shows that 95% of apps requesting less than 5 signature permissions, 65% of apps request 3 to 7 dangerous permissions, and 65% of apps request 5 to 10 normal permissions. Surprisingly, 35% of apps requested 5 to 20 customized permissions, and 15% of apps requested more than 20 customized permissions. This trend indicates a shift in the mobile programming paradigm from traditional Android libraries to reproducible third-party libraries.

Further analysis of customized permissions found that out of 3,212 customized permissions requested by 353 apps, there are 236 distinct permission names of this permission type. From permission names, there are 12 permissions embedded by more than 100 apps. Table 9 lists the top 20 requested third-party permissions used to support third-party libraries embedded by apps. Most of these permissions are useful in the push notification process from third-party cloud facilities to the device installing the app.

Another worth noting in apps’ permission analysis is the compliance of apps’ developers to the use of permission agreements and the

potential security risk to certain apps. We found that 26 apps request for `android.permission.MOUNT_UNMOUNT_FILESYSTEMS` (cf. Table 2). According to [10], this permission can be used to mount and unmount external storage such as SSD cards. However, this permission is only for pre-installed applications or applications that were already installed when the device was distributed. In addition, we discovered that 8 apps request `android.permission.DOWNLOAD_WITHOUT_NOTIFICATION` (cf. Table 2) which allows the app to download any file without user consent. This can be very dangerous if the app automatically downloads malicious applications into the device. `com.nexowallet` and `com.polehin.android` are two out of eight apps that requested this permission and those apps have been installed by more than 1 million devices and have review scores of 4.6 and 4.2 respectively.

Table 2. (Un)Mount file system and download without notifications permissions requested by the analysed apps.

Sensitive Permission	# of Apps (%)	Example App
<code>MOUNT_UNMOUNT_FILESYSTEMS</code>	26 (5.7%)	<code>com.bityard.us2</code>
<code>DOWNLOAD_WITHOUT_NOTIFICATION</code>	8 (1.8%)	<code>com.burency.app</code>

4.2 Third-party Libraries Penetration

We found a total of 59 distinct third-party libraries embedded in 391 apps. Due to the limited list in our dictionary and the obfuscation mechanism adopted by the apps, we were unable to detect third-party libraries in 66 apps ($\approx 14\%$ of 457 wallet apps). Depending upon their intended functionalities, we group the 59 distinct libraries into four main categories: Analytics, Ads & Tracker, Payment, and Social Media libraries.

Table 3. Dominant libraries grouped by the library category.

No	Analytics		No	Ads&Tracker	
	Names	Count (%)		Names	Count
1	Google Analytic	61 (3.7%)	1	Google Ads	259 (15.8%)
2	Mixpanel	20 (1.2%)	2	Appsflyer	66 (4.0%)
3	adjust	15 (0.9%)	3	Tencent	25 (1.5%)
4	Umeng	11 (0.7%)	4	AppLovin	4 (0.2%)
5	Flurry	10 (0.6%)	5	Appboy	4 (0.2%)
No	Payment		No	Social Media	
	Names	Count		Names	Count
1	Squareup	166 (10.1%)	1	Facebook	198 (12.1%)
2	Intuit	8 (0.5%)	2	Twitter	10 (0.6%)
3	Paypal	1 (0.1%)	3	Weibo	3 (0.2%)
4	Urbanairship	1 (0.1%)			

Table 3 shows the dominance of Google Analytics and Google Ads for the Analytics and Ads & Tracker categories which are embedded in 61 and 259 apps, respectively. As for the Payment and Social Media categories, Squareup and Facebook are the most popular and adopted in 166 and 198 apps, respectively.

According to data distribution, the adoption rate of third-party libraries by cryptocurrency wallet apps is considered to be low. Particularly, Table 4 shows that 172 (37.6%) apps did not adopt Ads & Tracker libraries whereas other 270 (59.1%) apps adopted only 1 or 2 libraries, 13 (2.8%) apps adopted from 3 to 5 libraries and 2 (0.2%) apps adopted more than 5 libraries. This adoption rate is much smaller than the adoption of Ads & Tracker libraries in other genres of apps, for example, measurement by Sentana et al., [36] found that 22.2% of Android health-related apps embed at least five different third-party libraries, or in the measurement results by [39] who found more than 43.0% of non-mobile health apps in their corpus embed more than 5 Ads & Tracker libraries. The two apps that embed the most Ads & Tracker libraries in their package are `network.xyocoin` and `com.callsfreecalls.android`, with 10 and 9 libraries, respectively and they have been installed more than 1 million times.

Table 4. Distribution of third-party libraries embedded by the analyzed apps.

# Of Libraries	Analytics	Payment	Social Media	Ads & Trackers
1	94 (20.6%)	166 (36.3%)	194 (42.5%)	211 (46.2%)
2	12 (2.6%)	5 (1.1%)	7 (1.5%)	59 (12.9%)
3	1 (0.2%)	0 (0.0%)	1 (0.2%)	10 (2.2%)
4	0 (0.0%)	0 (0.0%)	0 (0.0%)	2 (0.4%)
5	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (0.2%)
6+	0 (0.0%)	0 (0.0%)	0 (0.0%)	2 (0.4%)

The adoption rate of the other three types of libraries is less than the adoption rate of the Ads & Tracker library. There are only 202 (44.2%) apps that adopted 1 to 3 Social Media libraries, 171 (37.4%) adopted 1 to 3 Payment libraries, and only 107 (23.4%) apps adopted 1 to 3 Analytics libraries. This phenomenon reinforces the exposure made by [28] where advertising is not the main source of income in the cryptocurrency wallet ecosystem. Most apps support their operational costs through affiliate fees. In this model, the cryptocurrency wallet apps accommodate the crypto swap process from the cryptocurrency swap service provider. Some of the profits obtained by the service provider are from the difference in currency values which are shared with apps. Another business model is to make cryptocurrency wallets become a part of cryptocurrency exchange apps and get a split fee from the exchange transaction. In addition to the business model, some apps gain funding from the Initial Coin Offering (ICO) for cryptocurrency wallets that are integrated with certain coins.

Apart from the four types of third-party libraries, we also analyzed the existence of *suspicious* Cross-Library Data Harvesting (XLDH) libraries in

cryptocurrency wallet apps. We found that 13 (2.8%) apps embed XLDH libraries in their package as shown in Table 5. Particularly, `com.yandex.metrica` library, which is embedded by 4 apps, extracts the Google Advertising id, Android ID, and user MAC address and then exfiltrates the information to `https://startup.mobile.yandex.net`. Similar information was also extracted by `com.leanplum`, `cn.sharesdk`, `com.inmobi`. More detailed information is harvested by `com.umeng.socialize` which includes AccessToken and user data (ID/name/link/photo) on several social media platforms including Facebook, Twitter, Dropbox, Kakao, Yixin, Wechat, QQ, Sina, Alipay, Laiwang, Vk, Line, and LinkedIn.

Table 5. XLDH Libraries embedded in the analysed apps.

App ID	XLDH Library	Exfiltration Endpoint	# of Installs
<code>app.hodlify</code>	<code>com.yandex.metrica</code>	https://startup.mobile.yandex.net	1K+
<code>co.bitx.android.wallet</code>	<code>com.leanplum</code>	undetected	5M+
<code>com.evraon.trading</code>	<code>com.yandex.metrica</code>	https://startup.mobile.yandex.net	1K+
<code>com.ixx.android</code>	<code>cn.sharesdk</code>	http://api.share.mob.com/log4	5K+
<code>com.okinc.okex.gp</code>	<code>cn.sharesdk</code>	http://api.share.mob.com/log4	1M+
<code>io.bincap.exchange</code>	<code>com.yandex.metrica</code>	https://startup.mobile.yandex.net	500+
<code>network.xyocoin</code>	<code>com.inmobi</code>	https://sdkm.w.inmobi.com/user/e.asm	1M+
<code>com.btcc.mobiwallet</code>	<code>cn.sharesdk</code>	http://api.share.mob.com/log4	1K+
<code>com.zengo.wallet</code>	<code>com.leanplum</code>	undetected	100K+
<code>com.sixpencer.simplework</code>	<code>cn.sharesdk</code>	http://api.share.mob.com/log4	10K+
<code>com.fox.one</code>	<code>com.umeng.socialize</code>	http://plbslog.umeng.com/umpx_share	1K+
<code>com.beecrypt.beecrypthd</code>	<code>com.yandex.metrica</code>	https://startup.mobile.yandex.net	1K+
<code>com.hconline.iso</code>	<code>cn.sharesdk</code>	http://api.share.mob.com/log4	1K+

4.3 Malware Presences

Based on VirusTotal results, we found 25 apps detected containing malware. Specifically, 18 apps were detected by 1 antivirus engine, 3 apps were detected by 2 antivirus engines, and 4 apps were detected by more than 3 antivirus engines. Moreover, 9 antivirus engines detected malware on `com.top1.group.international.android`, 8 engines detected malware on `com.jex.trade`, and 7 and 4 engines detected malware on `com.legendwd.hyperpayW` and `im.token.app`, respectively.

We further cross-validate this analysis results with the anti-analysis measurement results in Sect. 4.4. As we shall show later in Sect. 4.4, there are four apps that embed the Jiagu packer to encrypt the .dex file. In this measurement, there are 4 antivirus engines (i.e., Ikarus, Fortinet, ESET-NOD32, and MaxSecure) that consistently detect the same malware in the four apps that embed *Jiagu* packer in their packages.

4.4 Anti-analysis detection

Figure 4 depicts our analysis results of anti-analysis methods employed by wallet apps. We discuss our analysis of the evasion methods in the following.

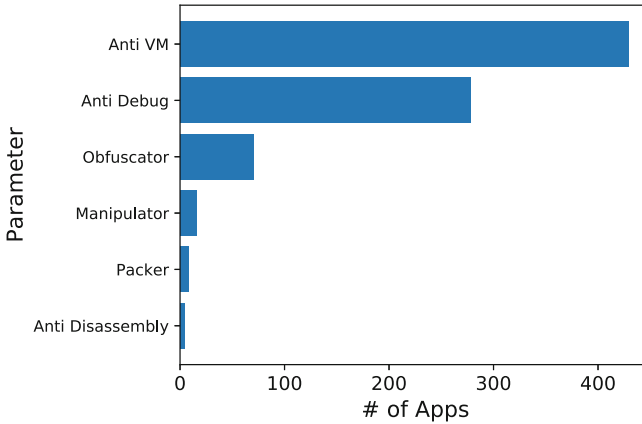


Fig. 4. Anti-Analysis mechanisms employed by the analysed wallet apps.

Manipulator. We found that 18 (3.9%) apps are marked as containing manipulator because the *Dalvik Executable (.dex)* files developed using *dexmerge* compiler. Note that *.dex* file, which exists in each APK, is a byte-code file converted from *Java.class*. Thus, it can be executed by the devices. Originally, *dex* file is developed using *dx* or *r8* compiler. APKiD tool will mark an app as containing a manipulator if (i) the original *.dex* files of the app are modified using a modification library such as *dexmerge* or (ii) *.dex* files are created from reverse-engineered source code using *dexlib* library, which is commonly used by decompiler tools such as *apktool* or *smali* [34,35]. APKiD tool identifies the manipulator by analyzing the change history in *Map Ordering Type* of the *.dex* files since the code sequence resulted from original *.dex* compiler, *dexmerge*, *dexlib* or *dex2lib* is different [13].

Anti Virtual Machine. We detected 429 (93.8%) apps adopting anti-virtual machine (Anti-VM) analysis in their package. An Anti-VM is a mechanism to detect whether the apps are executed on an emulator or real device. The goal is to impede the analysts to run the apps in an isolated environment such as Virtual Machine. The most common mechanism to check whether the device is emulated [30] is to analyze `build.prop` file containing a list of Build API methods, including `Build.Fingerprint`, `Build.Hardware`, and `Build.Device` and other device's properties. An alternative method is to check the *Telephony manager* which contains fixed API values for Android emulators including `getNetworkType()`, `getNetworkOperator()`, `getPhoneType()` and other network-related properties.

Anti Debug. We found 278 (60.8%) apps leveraging anti-debugging techniques to disrupt the reverse engineering of their source code. The anti-debugging technique ensures that the apps are not running under a debugger or changing the app's behavior when running under the debugger mode. Android provides two levels of debugging and anti-debugging protocol [29]. The first level of debugging can be conducted in communication protocol between Java Virtual Machine and debugger using Java Debug Wire Protocol (JDWP). We can identify anti-debugging by verifying if the setup includes the *debuggable flag* in *Application-Info* or by checking the *timer checks routine*. While the next level of anti-debug technique is to conduct traditional debugging by using *ptrace* in Linux *system call*. In this research, we found all of the 287 cryptocurrency wallets activating the debuggable flag of `Debug.isDebuggerConnected()` check, which is part of JDWP anti-debug level.

Obfuscator. Overall, we found 70 (15.3%) apps leveraging obfuscation tools in their package. Of that number, 3 apps obfuscated by Dexguard [14], 3 apps obfuscated by Arxan [11]. obfuscation is a process of concealing the original source code, binary code, or byte-code into an obscure set of characters, and 6 apps obfuscated by Clang [38]. Dexguard is a proprietary Android obfuscation tool that provides multi-layer protection against the static and dynamic analysis of byte-code, manifest, and all other resources included in distribution packages. While Dexguard obfuscated the byte-code level of Android Apps, Arxan and Clang are categorized as Low-Level Virtual Machines tools that obfuscated the binary code level of Android Apps.

Anti Disassembly. This technique prevents the reverse engineer from disassembling the byte-code into higher-level code such as Java or Smali. The most popular anti-disassembly mechanism in Android is by developing part of the code segment in C or C++ using Native Development Kit (NDK) [37]. NDK provides platform libraries to manage native activities and access physical device components [7]. NDK uses *CMake* as a native library compiler that creates a different *byte-code* structure compared to the code written in Java or Kotlin. Hence, it impedes common Android tools such as Apktool or Smali to disassemble the *byte-code*. It required an advanced reverse engineer familiar with ARM processor architecture, Assembler language, Java Native Interface (JNI) convention, and Application Binary Interface (ABI) compiler to decompile the *byte-code*. More advanced techniques are used by malware developers to evade disassembly tools explained by [27]. The technique leveraging *jmp* and *call* commands in *byte-code* level to direct the instruction flow to a certain location with a constant value, or direct the flow to the same target memory location. This technique will produce a false listing of source code when it disassembles using a decompiler tool. We found 4 apps leveraging the anti-disassembly techniques. Analysis of those apps returns the value of "Illegal class name", indicating the decompiler result violates the standard structure of Java or Kotlin.

Packer. Initially, Packer was created to protect intellectual property in the form of source code on Android apps. These tools prevent third-parties from analyzing source code or doing reverse engineering. Packer works by encrypting the `.dex` files in the Android package and storing the encryption results in a secure new block architecture. Unlike the obfuscator which will be decrypted when executed on the device, the packer is remained stored in the packer block and uses ununpacker when it will be executed on the device. However, commercial packers are often used by malware developers to hide malicious codes [5, 12]. We found that 7 (1.5%) apps use packers, of which 4 apps embed `Jiagu` packer, while the three remaining apps use `Ijiami`, `SecNeo`, and `Yidun` packers. Note that the wallet apps using `Jiagu` packer are also detected as malware by VirusTotal (cf. Sect. 4.3).

5 Dynamic Analysis Result

We investigate the runtime, network behavior of the cryptocurrency wallet apps. Out of 457 tested apps, we successfully captured the network traffic of 391 apps. The reasons that we failed to capture traffic of 66 apps include the app navigation process crashing and the app installation failure.

5.1 Securing Network Requests

To secure communication from in-path attackers, apps leverage transport layer security (TLS) and HTTPS. For each wallet app, we filter TCP flows in network traces (saved in `pcap` file), to identify whether the connection establishment between the app and the Internet is over TLS. We also identify the TLS version, cipher name, and the HTTP version used in the apps' traffic. We also filter network traffic for HTTP(S) requests. We found that 261 (66.7%) apps have established *TLS* for all of their traffic where they use *TLSv1.2* and/or *TLSv1.3* in their traffic, 148 (37.8%) apps include traffic that was not over *TLS* when they use *HTTP* scheme in their traffic and 9 (2.3%) apps do not use *TLS* in all of their traffic. Of the 391 apps that we successfully captured network traffic, 148 (37.8%) apps transmitted some of their traffic over unencrypted HTTP protocol, which could be vulnerable to modification during the transmission and be transparent for traffic analysis.

5.2 Sensitive Data Aggregation and Sharing

We observe whether an app collects and transmits data related to users' information and wallet information in their network traffic. Based on the captured traffic dumped into the `.har` file, we extract the HTTP request package's header and payload and find whether user information is in the app's traffic.

Data Aggregation Practices. For each app, we extract all HTTP requests with the POST method and inspect the *postData* field of the package’s header to identify data related to the user, credential, device, location, and wallet information. If so, the app is claimed to collect personal information as well as wallet information. In order to detect if an app collects user information, we match predefined keywords with the key values in the *postData* field. For any matching keywords, we claim that the app collects user information. A similar strategy is applied to the detection of collecting device information, credential information, location information, and wallet information. To define related keywords to each information category, we first scan all unique keys in the *postData* field of all apps’ traffic. We then search for related keywords for each information category. As the keys used in *postData* field to refer to the same information are not exactly matched. We carry all possible keywords for each information category to ensure that we do not miss any app that collects a piece of given information. For instance, the predefined keywords for user information are determined based on the list of all unique keywords we scanned for from all apps’ traffic. We search for all keys in that list that contain the string ‘*name*’. After that, we manually filter out keywords that include ‘*name*’ but do not refer to personal information. Finally, we obtain a predefined keyword list related to user information including ‘*first_name*’, ‘*last_name*’, ‘*surname*’, ‘*Username*’, ‘*email*’, ‘*account_name*’, ‘*user_name*’, ‘*name*’, ‘*partner_name*’, ‘*given_name*’, ‘*named_user_id*’, ‘*user_name*’, ‘*full_name*’, ‘*emailAddress*’, ‘*emailToken*’, ‘*email_id*’, ‘*fullname*’. Similarly, we obtain the keywords for device information, credentials, location, and wallet information. We provide details of our keywords for each user-related information category in Appendix B.

Based on the apps’ traffic and our predefined keywords, we identify a number of apps that collect personal information as well as wallet information. Table 6 shows the identified number of apps (out of 391 apps) that collect a given information.

Table 6. Number of apps collecting personal and wallet information.

Collected Attributes	Apps Collecting Info	Apps Sharing with FP	Apps Sharing with TP	Sharing via HTTP
User Info	67 (17.1%)	10 (2.5%)	57 (14.6%)	4 (1.0%)
Credential Info	50 (12.8%)	9 (2.3%)	41 (10.5%)	4 (1.0%)
Device Info	83 (21.2%)	5 (1.2%)	78 (19.9%)	5 (1.3%)
Location Info	16 (4.1%)	-	16 (4.1%)	-
Wallet Info	44 (11.2%)	2 (0.5%)	42 (10.7%)	-

Sharing Sensitive Data with Third-Parties. For all apps that collect personal information and wallet information, we further inspect the requested URL in the *URL* field of the packet’s header to identify if the collected data is shared with third parties.

We also identified apps that share the collected information with TPs via the HTTP scheme. Table 6 also shows the identified number of apps that share collected personal information and wallet information with third parties and apps that share these data using the HTTP scheme. Specifically, we found that 15 apps (e.g., `com.btcc.mobiwallet` and `com.coinburp.mobile`) share user credentials and device information with third-parties via HTTP.

5.3 Third-party Domain Request

We identify third-party domains in the app traffic from the captured pcap file. For each analysed app, we capture all requested domains from har file and classify requested domains into first-party domains and third-party domains (i.e., Domains that do not belong to apps' developers). To determine apps' communication with third-parties, we leveraged filter lists: EasyList [1] an advert block list, and EasyPrivacy [2] a supplementary block list for tracking, to filter advert and tracking related third-party domains requested by the tested apps.

113 (28.9%) apps requested more than 10 unique domains during the app execution. Among them, the app with the handle name of `io.atomicwallet` requested 66 different domains. Individually, 70 (17.9%) apps requested *one* third-party domain, 338 (86.4%) apps requested more than *two* third-party domains and 72 (18.4%) apps requested more than 10 third-party domains. The app with the handle name of `com.paymintlabs.paymint` requested 44 different third-party domains. Top 5 third-party domains `firebaseinstallations.googleapis.com`, `google.com`, `crashlytics.com`, `rqmob.com` and `facebook.com` were found to be requested by 169 (43.2%), 133 (34%), 91 (23.2%), 87 (22.2%), and 76 (19.4%) apps, respectively.

6 Privacy Policy Compliance and User Review Results

6.1 Privacy Policy Analysis

Upon the privacy policies extraction, we managed to extract 327 privacy policies and failed to extract 130 privacy policies due to several reasons as listed in Table 10. 31 (6.8%) cryptocurrency wallet apps provide an untraceable link for their privacy policy in Play Store content page information, while 17 (3.7%) cryptocurrency wallet apps did not provide any link to their privacy policy documents. The result from privacy policy extraction is then used in the following analysis:

Compliance to Provide Privacy Policy Link. As a part of our first analysis, which is related to the developer compliance to provide a privacy policy link, we used the extraction result and exclude several reasons for failures, such as HTTP error 403 and HTTP error 503 which is used by the server as a part of the defense against Denial of Service (DoS) attack (cf. Table 10). We also exclude several links that can be traced if it navigated manually by humans to

reach privacy policy such as the HTTP error 104 and HTTP error 111 as well as SSL_ERROR_BAD_CERTIFICATE_DOMAIN. And for the last, we exclude the apps that are no longer available in Play Store. Hence, in total Overall, we found 87 (19.0%) cryptocurrency wallet apps failed to provide a privacy policy on Google Play.

App’s Adherence To Information Sharing Policy. This analysis consists of two steps: classification of the app’s privacy policy for collecting user information and validation of sharing the collected information with third-parties (TP). For each app, we first classify the app’s privacy policy as *true* or *positive* if they declare or claim that the apps share the user information with the TPs, while the app’s privacy policy is labeled as *false* or *negative* if they do not declare to share the user information to the TPs. Second, we cross-validate the apps that are labeled to be false to the third-party libraries embedded by corresponding apps and cross-validate those apps to the third-party domains accessed during the app’s runtime. We assume that the cryptocurrency wallet apps violate the privacy policy in terms of sharing user information with the third-parties if they do not declare it in privacy policies while embedding tracking libraries (Sect. 4.2) or they send user information to the third-party domains (Sect. 5.3).

To automatically detect if the app claims to share user information with TPs, we rely our classification on an existing corpus of privacy policies [46] annotated by legal experts. This corpus contained 213 and 137 privacy policies labeled as positive and negative, respectively. However, the positive annotation in this corpus is only given for the policy that contains very specific information sharing and annotates the general information sharing phrases to the negative. For instance, specific phrases such as “We share your email address with third parties” would be annotated as *positive*, while general phrases such as “We share your information with third parties” would be labeled as *negative*. Hence, we re-label this corpus by removing privacy policies that contain general information sharing in the negative class. After this step, 35 privacy policies remain in the negative subset. To avoid an imbalance of class data, we reduce the size of the positive subset and make our *new corpus* containing 68 and 35 privacy policies labeled as positive and negative, respectively. We then train a Support Vector Machine (SVM) classification model based on our new corpus.

Since our classification model is trained in English-based language, we then exclude non-English privacy policies from 327 (out of 457) cryptocurrency wallet apps from which we could extract their privacy policy text. We found that 60 (out of 327) privacy policies were written in a non-English language, including binary texts. We then classify these 267 privacy policies using our trained classification model. As a result, we found 93 privacy policies classified as false meaning that the corresponding apps did not declare or claim to share user information with TPs, and 175 privacy policies classified as true indicating that the apps declare to share user information with the TPs.

We then cross-validate 93 apps which corresponding privacy policy apps labeled as false to the apps adopting third-party libraries in Sec. 4.2. We then marked the apps that embed at least one third-party library as apps that failed

to adhere to privacy policies related to sharing users' information with third parties. As a result, we found 78 (17.0%) cryptocurrency wallet apps fall into this category.

We examine 93 apps that do not claim to share user information with third parties (i.e., being labeled as false in the privacy policy classification) by investigating their network traffic during execution time (Sect. 5.2). We found 26 apps that did not declare to share user information with the TPs had shared user information including personal information, device information, credentials, location, and wallet information. Details of our results are shown in Table 7. Consequently, we also marked these apps as violating privacy policy in terms of sharing user information with TPs.

Table 7. Apps sharing user information with TPs w/o claiming in privacy policy

Attribute	# Apps	Shared Information
Personal Info	4 (4.3%)	'username', 'surname', 'email', 'name'
Device Info	11 (11.8%)	'deviceType', 'device uuid', 'deviceData', 'device name' 'deviceFreeSpace', 'device id', 'device token', 'deviceModel'
Credentials Info	4 (4.3%)	'password', 'password score'
Location Info	2 (2.2%)	'locale'
Wallet Info	5 (5.3%)	'primary public key', 'branch key', 'wallet version' 'backup public key', deployment key'

6.2 User Reviews Analysis

We use users' negative comments (reviews) to capture the perceptions and concerns about the security and privacy features of wallet apps. Our reasoning to focus our analysis on negative reviews, 1- and 2-star reviews appearing on Google Play, for popular apps is that any serious concern exposed by a user should receive a negative review. Overall, we obtained 673,424 reviews corresponding to 403 (88.2%) wallet apps. We noted that 54 (12.8%) apps have no user reviews while 78.5% (359) of the analyzed apps have at least one negative comment with a total of 175,564 or 26.1% of total reviews.

For further analysis, we process negative reviews by grouping them based on the type of complaint. We detect the complaint type in each negative review based on the occurrence of keywords in the review and group it into six categories, including fraudulent, bugs, authentication, security, usability, ads, and tracker. For example, the appearance of the word "scam", "fake" or "liar" in a negative review indicates a complaint about **fraudulent** activities. Similarly, reviews containing keywords such as "bugs" or "error" into **Bugs complaint**. More details about the complaint category and the keywords mapping can be seen in Table 8.

Table 8. Distribution of users’ complaints in different categories. The percentage of users’ complaints is measured by dividing “# of Complaints” using the total negative reviews.

Category	# of Complaints (%)	#of Apps(%)	Keywords
Fraudulent	50,628 (28.8%)	287 (62.8%)	‘scam’, ‘fake’, ‘money’, ‘liar’, ‘purchase’, ‘payment’, ‘credit card’, ‘debit card’, ‘cash’, ‘manipulation’
Bugs	22,742 (13.0%)	256 (56.0%)	‘bug’, ‘error’, ‘crash’, ‘update’, ‘upgrade’, ‘not responding’, ‘freeze’, ‘stuck’
Authentication	37,791 (21.5%)	262 (57.3%)	‘verification’, ‘verify’, ‘verif’ ‘verified’, ‘account’, ‘notification’, ‘login’, ‘register’
Security	4,692 (2.7%)	176 (38.5%)	‘security’, ‘secure’, ‘hack’, ‘bot’ ‘hacking’, ‘hacker’, ‘insecure’
Usability	11,599 (6.6%)	213 (46.6%)	‘confuse’, ‘confusing’, ‘bad’, ‘rubbish’, ‘slow’, ‘junk’, ‘user interface’
Ads and tracker	772 (0.4%)	102 (22.3%)	‘ads’, ‘video ads’, ‘tracker’, ‘intrusive ads’, ‘massive ads’ ‘advert’, ‘advertisement’

Table 8 also shows the number of complaints and their percentage of total negative reviews grouped by category. Most of the users complaints are related to *fraud* with 50,628 (22.5%) complaints, followed by *authentication* with 37,791 (14.9%), *bugs* with 22,742 (12.3%), *usability* with 11,599 (4.5%), *security* with 4,692 (2.6%), and *ads and tracker* with 772 (0.6%). This result is surprising as we expect ads and trackers to get a large proportion of complaints considering that the corpus of the analyzed wallets apps are *free* apps. However, after looking in more detail at the business process of wallets and considering the low adoption of third-party ads and tracker libraries discussed in Sect. 4.2, it is acceptable that ads and trackers are not a “big player” in wallet apps. To observe the distribution of complaint categories in cryptocurrency wallet apps, we calculate the ratio of complaints occurrence per category to the total negative reviews for each app. This is to accommodate gaps between the apps with a large number of reviews and a small number of reviews. For example, *com.coinbase.android* has a fraudulent ratio of 32.8% calculated from 10,210 fraudulent complaints divided by 31,107 negative reviews owned by the app.

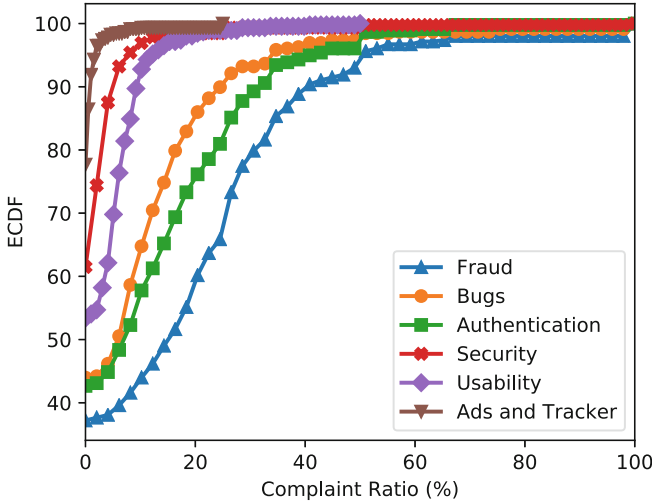


Fig. 5. ECDF of user complaints ratio per app across various categories.

Based on this calculation, fraud still dominates compared to other complaint categories. There are 20 apps (4.4%) that have more than 50% fraud complaints. All of these apps have a totally negative review below 100 and 15 apps have a total review (positive, normal, and negative) under 100. Figure 5, depicts the distribution of the ratio of negative reviews to the total number of reviews across various categories.

We also cross-validated the negative review ratio of VirusTotal’s detection results on malicious apps, as shown in Table 11. As a result, of the 25 apps that were detected as malicious by the VirusTotal, 8 apps have received negative ratios above 25% where 3 of them have been installed more than 100K times.

7 Related Work

Recently, few work studied the security and privacy risks of Android cryptocurrency wallet apps. For instance, He et al. [16] discovered threat vectors of cryptocurrency wallet apps and studied the security weaknesses of both the Android system and cryptocurrency wallets. However, this work provides an assessment of just two apps.

Hu et al. [19] explored the security features of the 10 most popular Bitcoin wallet applications and discovered three security vulnerabilities of Bitcoin wallet applications including privacy leakage, spamming and financial loss and demonstrated corresponding proof-of-concept attacks. Sai et al. [33] examined the security issues of 48 commonly used Android cryptocurrency wallet applications. Nevertheless, this work mainly focuses on security vulnerabilities rendered by requested permissions. Li et al. [23] studied the security risks of 8 common

Android cryptocurrency wallet apps by examining requested permissions, backup files, Android clipboard, and accessibility service.

Biryukov and Tikhomirov [4] proposed a security and privacy analysis method based on clustered transactions on timing analysis for Bitcoin, Dash, Monero, and Zcash cryptocurrencies. The measurement relies on four main parameters including information Leak to external storage, XSS attacks via Javascript in WebView, Insecure connection, and information leak into logs. Uddin et al. [41], proposed Horus, a security assessment framework for Android Cryptocurrency apps that includes static and dynamic analysis. Static analysis in Horus focuses on the security assessment based on the API calls while dynamic analysis was only used to profile the app's structure and to identify the key location storage during the runtime.

In contrast, our work provides a large-scale comprehensive analysis of the security and privacy features of 457 Android cryptocurrency wallet applications. The static analysis in our study consists of several security and privacy parameters that include permission, app packaging, anti-analysis adoption, the use of third-party libraries, and malware presence. While the dynamic analysis in our study conducted the identification of standard security practices and capture the network traffic to seek the potential information leakage to the third party. This study also provides analysis results from the user's point of view and potentially of developer compliance related to the privacy policy.

8 Conclusion

Unlike most free apps which use ads to monetize their apps, wallet apps use split or sharing fees to support their business process. Hence, it gives a unique characteristic to the apps' structure in terms of third-party libraries' adoption and permission requests. In addition, most wallet apps also utilize third-party cloud infrastructure, especially to support push notification facilities, which causes the adoption of third-party libraries in this regard to be large. The adoption of this third-party library is helpful in accelerating application development, but on the other hand, it also leaves security and privacy issues for users.

In this study, we found several permission requests that are very dangerous for users and also the adoption of a malicious library adopted by wallet apps. The adoption of anti-analysis techniques by wallet apps has also led to a malware-detecting issue by a number of antivirus engines. We also found several violations of the privacy policy agreement between the developer and the user by the wallet apps. Thus, we expect the results of this assessment would assist users in observing the security and privacy of wallet apps on marketplaces such as the Google Play Store. To foster future research, we release our code and dataset used in this paper to the research community: <https://walletapps2021.github.io/>.

Acknowledgements. The first author is an Indonesian Endowment Fund for Education (LPDP) scholarship awardee with the ID 20193221014024. Part of this research

is funded by Macquarie University Higher Degree Research (HDR) support fund and supported by Macquarie University Cyber Security Hub.

A List of Anti-analysis Methods in Wallet Apps

We further explain the anti-analysis techniques used by the analyzed wallet apps.

- **Manipulator.** It is a tool or mechanism to modify .dex file and to re-package the modified .dex file into new apps [13,34,35].
- **Anti Virtual Machine.** A technique [24,31,42] to detect and evade analysis via sandboxes Android emulators or virtual machine.
- **Anti Debug.** It is a mechanism to prevent program analysis or debugging activities [29].
- **Obfuscator.** Obfuscation is a process of concealing the original source code, binary code, or byte-code into an obscure set of characters by encrypting or changing the original code without omitting the functionality. The goal is to impede the reverse engineering of the code by unauthorized parties. An *Obfuscator* renames libraries, variables, methods and class names [8,14].

Table 9. Top 20 customized and third-party permissions requested by the analyzed apps.

No	Permission Name	# of Request (%)
1	com.google.android.c2dm.permission.RECEIVE	307 (67.2%)
2	com.google.android.c2dm.permission.SEND	297 (65.0%)
3	.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE	290 (63.5%)
4	.gms.auth.api.signin.permission.REVOCATION_NOTIFICATION	113 (24.7%)
5	com.huawei.android.launcher.permission.CHANGE_BADGE	102 (22.3%)
6	com.anddoes.launcher.permission.UPDATE_COUNT	101 (22.1%)
7	com.sonyericsson.home.permission.BROADCAST_BADGE	101 (22.1%)
8	com.sec.android.provider.badge.permission.WRITE	101 (22.1%)
9	com.sec.android.provider.badge.permission.READ	101 (22.1%)
10	com.majeur.launcher.permission.UPDATE_BADGE	101 (22.1%)
11	com.htc.launcher.permission.READ_SETTINGS	101 (22.1%)
12	com.htc.launcher.permission.UPDATE_SHORTCUT	101 (22.1%)
13	com.huawei.android.launcher.permission.READ_SETTINGS	96 (21.0%)
14	com.huawei.android.launcher.permission.WRITE_SETTINGS	96 (21.0%)
15	com.sonymobile.home.permission.PROVIDER_INSERT_BADGE	96 (21.0%)
16	android.permission.READ_APP_BADGE	92 (20.1%)
17	com.oppo.launcher.permission.READ_SETTINGS	91 (19.9%)
18	com.oppo.launcher.permission.WRITE_SETTINGS	91 (19.9%)
19	me.everything.badger.permission.BADGE_COUNT_READ	90 (19.7%)
20	me.everything.badger.permission.BADGE_COUNT_WRITE	90 (19.7%)

Table 10. Cause of Privacy Policy Extraction Failed.

No.	Caused of Failed	# of Apps (%)
1	404: Not Found	31 (6.8%)
2	403: Forbidden	19 (4.2%)
3	No Privacy policy link	17 (3.7%)
4	3: Temporary failure in name resolution	14 (3.1%)
5	2: Name or service not known	10 (2.2%)
6	No Metadata Found	9 (2.0%)
7	503: Service Temporarily Unavailable	7 (1.5%)
8	110: Connection timed out	3 (0.7%)
9	SSL: CERTIFICATE_VERIFY_FAILED	3 (0.7%)
10	522: Cloudflare times out	3 (0.7%)
11	SSLERROR_BAD_CERT_DOMAIN	2 (0.4%)
12	502: Bad Gateway	1 (0.2%)
13	104: Connection reset by peer	1 (0.2%)
14	526: Origin SSL Certificate Error	1 (0.2%)
15	403: Ip Forbidden	1 (0.2%)
16	500: Internal Server Error	1 (0.2%)
17	400: Bad Request	1 (0.2%)
18	308: Permanent Redirect	1 (0.2%)
19	307: server returned infinite loop	1 (0.2%)
20	111: Connection refused	1 (0.2%)
21	0: Error	1 (0.2%)
22	5: No address associated with hostname	1 (0.2%)
23	UNRECOGNIZED_NAME_ALERT	1 (0.2%)
	Total	130(28.4%)

- **Anti Disassembly.** It is a technique aiming to prevent the extraction of symbolic representations of the assembly code instructions from the APK of an Android app [18, 22].
- **Packer.** It is a techniques aiming at evading reverse engineering by encrypting the `.dex` file [5, 12].

B Keywords for Personal Data Transmission Analysis

Device information keywords: *deviceDescription, unidentified_device, deviceID, deviceToken, deviceRegKey, rooted_device, device_model, device_audio, device-Data, device_token, deviceType, device_brand, deviceId, devicetoken, kochava_device_id, device_name, deviceFreeSpace, deviceOEM, deviceOS, device, device_uuid, device_os, deviceHeight, device_api, deviceInfo, device_type, device_time, deviceWidth,*

actual_device_type, deviceRm, deviceKey, deviceModel, device_fingerprint_id, device_id, deviceMAC, deviceId, deviceVersion, deviceFingerPrintId, device_data

Credential information keywords: *second_password_confirmation, password, password_score, password_confirmation, second_password, password*

Location information: *localeIdentifier, location, locale_language_code, locale_country_code, locale, h_region*

Collected wallet information: *devkey, walletId, previous_deployment_key, walletAddress, key_index, subwallets, coin, master_public_key, deployment_key, wallet_type, deployment_key, gApikey, wallet_version, bitcoin_notification_enabled, key, sort_key, previous_deployment_key, temporary_private_key, primary_public_key, branch_key, public_key, walletAddressIndex, wallet_transaction_notification_enabled, wallets, coinId, backup_public_key, wallet_id, walletAddressNm, api_key, walletID, pub_key*

Table 11. List of cryptocurrency wallet apps that are considered as malicious by users in Google Play reviews and by VirusTotal (AV-positive column). For each cryptocurrency wallet app, the NR-Ratio represents the ratio of the number of negative users’ comments to the total number of all users’ comments.

#	App ID	NR-Ratio	Installs	Rating	AV-positives
1	com.top1.group.international.android	1.0%	10000+	4.7	9
2	com.jex.trade	43.1%	500000+	4.1	8
3	com.legendwd.hyperpayW	10.4%	50000+	4.8	7
4	im.token.app	25.2%	500000+	4.1	4
5	com.vidulumwallet.app	13.8%	1000+	4.2	2
6	com.remint.app	18.8%	10000+	4.4	2
7	com.portto.blocto	23.8%	10000+	4.3	2
8	roseon.finance	4.6%	10000+	4.8	2
9	com.fox.one	2.9%	1000+	4.8	1
10	com.studentcoin	63.5%	50000+	4.4	1
11	one.mixin.messenger	15.8%	10000+	4.5	1
12	net.ethylte.com	2.9%	5000+	4.5	1
13	augstrain.asn	33.3%	10+	0.0	1
14	co.edgesecure.app	27.1%	100000+	4.0	1
15	com.viabtc.pool	14.8%	100000+	4.6	1
16	com.bitcoinglobal	50.0%	5000+	0.0	1
17	com.crypto.multiwallet	18.1%	100000+	4.4	1
18	com.friendst.strangr	80.0%	10000+	3.8	1
19	com.quidax.app	32.4%	100000+	3.7	1
20	com.digifinex.app	21.0%	100000+	3.9	1
21	com.lingxi.bexplus	1.8%	50000+	4.7	1
22	app.goodcrypto	15.5%	100000+	4.5	1
23	com.enjin.mobile.wallet	9.5%	500000+	4.6	1
24	com.holacoins.wallet	20.4%	50000+	4.2	1
25	africa.bundle.mobile.app	9.9%	100000+	4.5	1

References

1. Easylist (2021). <https://easylist.to/easylist/easylist.txt>
2. Easyprivacy. <https://easylist.to/easylist/easyprivacy.txt> (2021)
3. BBVA: Blockchain - what are the differences between a digital currency and a cryptocurrency? (2021). <https://www.bbva.com/en/what-are-the-differences-between-a-digital-currency-and-a-cryptocurrency/>
4. Biryukov, A., Tikhomirov, S.: Security and privacy of mobile wallet users in bitcoin, dash, monero, and zcash. *Pervasive and Mobile Computing* **59**, 101030 (2019). <https://doi.org/10.1016/j.pmcj.2019.101030>. <https://www.sciencedirect.com/science/article/pii/S1574119218307181>
5. Caijun, S., Hua, Z., Sujuan, Q., Nengqiang, H., Jiawei, Q., Hongwei, P.: DexX: a double layer unpacking framework for android. *IEEE Access* (2018)
6. Chau, N., Jung, S.: An entropy-based solution for identifying android packers. *IEEE Access* (2019)
7. Developer, A.: The android NDK: toolset that lets you implement parts of your app in native code, using languages such as C and C++. (2020). <https://developer.android.com/ndk>
8. Developer, A.S.: Shrink, obfuscate, and optimize your app (2020). <https://developer.android.com/studio/build/shrink-code>
9. Developers, A.: Permissions overview: android developers (2020). <https://developer.android.com/guide/topics/permissions/overview>
10. Developers, A.: Android documentation - manifest.permission (2021). https://developer.android.com/reference/android/Manifest.permission#MOUNT_UNMOUNT_FILESYSTEMS
11. Digital.ai: Arxan: App code obfuscation. <https://digital.ai/glossary/app-code-obfuscation> (2020). Accessed 18 Aug 2021
12. Duan, Y., et al.: Things you may not know about android (un)packers: a systematic study based on whole-system emulation. In: *NDSS* (2018)
13. Fenton, C.: Building with and detecting android's jack compiler (2016). <https://calebfenton.github.io/2016/12/01/building-with-and-detecting-jack/>
14. Guardsquare-Mobile-Application-Protection: Dexguard: Android app security - protecting android applications and sdks against reverse engineering and hacking (2020). <https://www.guardsquare.com/en/products/dexguard>
15. Hashmi, S.S., Ikram, M., Smith, S.: On optimization of ad-blocking lists for mobile devices. In: *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pp. 220–227 (2019)
16. He, D., et al.: Security analysis of cryptocurrency wallets in android-based applications. *IEEE Network* **34**(6), 114–119 (2020)
17. He, R., et al.: Beyond the virus: a first look at coronavirus-themed mobile malware (2020)
18. Hsieh, W.C., Engler, D.R., Back, G.: Reverse-engineering instruction encodings. In: *USENIX Annual Technical Conference, General Track*, pp. 133–145 (2001)
19. Hu, Y., et al.: Security threats from bitcoin wallet smartphone applications: vulnerabilities, attacks, and countermeasures. In: *ACM CODASPY* (2021)
20. Ikram, M., Kaafar, M.A.: A first look at mobile ad-blocking apps. In: *IEEE NCA* (2017)
21. Ikram, M., Vallina-Rodriguez, N., Seneviratne, S., Kaafar, M.A., Paxson, V.: An analysis of the privacy and security risks of android VPN permission-enabled apps. In: *ACM IMC* (2016)

22. Kruegel, C., Robertson, W., Valeur, F., Vigna, G.: Static disassembly of obfuscated binaries. In: USENIX security Symposium, vol. 13, p. 18 (2004)
23. Li, C., He, D., Li, S., Zhu, S., Chan, S., Cheng, Y.: Android-based cryptocurrency wallets: attacks and countermeasures. In: 2020 IEEE International Conference on Blockchain (Blockchain), pp. 9–16. IEEE (2020)
24. Maier, D., Müller, T., Protsenko, M.: Divide-and-conquer: why android malware cannot be stopped. In: 2014 Ninth International Conference on Availability, Reliability and Security, pp. 30–39. IEEE (2014)
25. matlink: Google play downloader via command line v3.25. <https://github.com/matlink/gplaycli> (2018). Accessed 10 Oct 2020
26. mitmproxy: - an interactive https proxy. <https://mitmproxy.org> (2020)
27. Nair, R.: Techbliss - tutorial anti-disassembly techniques used by malware (a primer) (2015). <https://www.techbliss.org/threads/anti-disassembly-techniques-used-by-malware-a-primer-by-rahul-nair.804/>
28. Ohayon, O.: Hackernoon - the business model of crypto-wallets (2018). <https://hackernoon.com/the-business-model-of-crypto-wallets-89aeed8322dc>
29. OWASP: testing anti-debugging detection (mstg-resilience-2) - android anti-reversing defenses (2020). <https://mobile-security.gitbook.io/mobile-security-testing-guide/android-testing-guide/0x05j-testing-resiliency-against-reverse-engineering> oWASP Mobile Security Guide - Accessed: 18/01/2020
30. OWASP: testing emulator detection (mstg-resilience-5) - android anti-reversing defenses (2020). <https://mobile-security.gitbook.io/mobile-security-testing-guide/android-testing-guide/0x05j-testing-resiliency-against-reverse-engineering>. oWASP Mobile Security Guide - Accessed 18 Jan 2020
31. Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., Ioannidis, S.: Rage against the virtual machine: hindering dynamic analysis of android malware. In: Proceedings of the Seventh European Workshop on System Security, pp. 1–6 (2014)
32. RedNaga: ApkId - anti-analysis open source tools (2016). <https://github.com/rednaga/APKiD>
33. Sai, A.R., Buckley, J., Le Gear, A.: Privacy and security analysis of cryptocurrency mobile applications. In: 2019 Fifth Conference on Mobile and Secure Services (MobiSecServ), pp. 1–6. IEEE (2019)
34. Security, R.: APKiD and android compiler fingerprinting (2016). https://rednaga.io/2016/07/30/apkid_and_android_compiler_fingerprinting/
35. Security, R.: Detecting pirated and malicious android apps with apkid (2016). https://rednaga.io/2016/07/31/detecting_pirated_and_malicious_android_apps_with_apkid/
36. Sentana, I.W.B., Ikram, M., Kaafar, M., Berkovsky, S.: Empirical security and privacy analysis of mobile symptom checking apps on google play. In: Proceedings of the 18th International Conference on Security and Cryptography - SECRYPT (2021)
37. SIMFORM: How to avoid reverse engineering of your android app? (2015). <https://www.simform.com/how-to-avoid-reverse-engineering-of-your-android-app/>
38. Source, G.: Android clang/llvm prebuilts (2020). <https://android.googlesource.com/platform/prebuilts/clang/host/linux-x86/+master/README.md>. Accessed 18 Aug 2021
39. Tangari, G., Ikram, M., Ijaz, K., Kaafar, M.A., Berkovsky, S.: Mobile health and privacy: cross sectional study. *BMJ* **373**, n1248 (2021). <https://doi.org/10.1136/bmj.n1248>. <https://www.bmj.com/content/373/bmj.n1248>

40. Tangari, G., Ikram, M., Sentana, I.W.B., Ijaz, K., Kaafar, M.A., Berkovsky, S.: Analyzing security issues of android mobile health and medical applications. *J. Am. Med. Inform. Assoc.* **28**(10), 2074–2084 (2021)
41. Uddin, M.S., Mannan, M., Youssef, A.: Horus: a security assessment framework for android crypto wallets. In: Garcia-Alfaro, J., Li, S., Poovendran, R., Debar, H., Yung, M. (eds.) *SecureComm 2021*. LNICTST, vol. 399, pp. 120–139. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90022-9_7
42. Vidas, T., Christin, N.: Evading android runtime analysis via sandbox detection. In: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, pp. 447–458 (2014)
43. VirusTotal: Multitude anti-virus engines (2021). <https://www.virustotal.com/gui/home/upload>
44. Wang, J., et al.: Understanding malicious cross-library data harvesting on android. In: *USENIX Security Symposium* (2021)
45. Xia, P., et al.: Characterizing cryptocurrency exchange scams. *Comput. Secur.* **98**, 101993 (2020)
46. Zimmeck, S., et al.: Maps: Scaling privacy compliance analysis to a million apps. *Proceed. Priv. Enhan. Technol.* **2019**(3), 66–86 (2019)

Author Index

A

Acharya, Anasuya II-611
Alnahawi, Nouri II-85
Andreeva, Elena II-3
Aragon, Nicolas I-725
Asghar, Hassan Jameel I-284
Ashur, Tomer II-611
Atapoor, Shahla II-219
Atkins, Conor I-284

B

Baghery, Karim II-219
Banik, Subhadeep I-178
Bao, Yue II-115
Bardi, Sara I-309
Becher, Kilian I-451
Beguinot, Hugo II-516
Bemmann, Pascal II-460
Berndt, Sebastian II-460
Biryukov, Alex I-149

C

Cardoso dos Santos, Luan I-149
Cascudo, Ignacio II-645
Chan, T.-H. Hubert II-582
Chen, Chuanxi I-237
Chen, Hongbin I-583
Chen, Siwei I-119
Cheriere, Agathe I-725
Chevalier, Céline I-634, II-516
Chi-Domínguez, Jesús-Javier II-276
Cho, Jin-Hee I-555
Chou, Kai-Hsiang I-260
Chow, Sherman S. M. II-366, II-484
Ciampi, Michele II-555
Cohen, Efrat II-611
Collins, Daniel I-178
Cong, Kelong II-248
Conti, Mauro I-309
Cozzo, Daniele II-219

D

Dang, Ziqin I-476
Davi, Lucas II-32
David, Bernardo II-645
Delavignette, Gaetan I-425
Deo, Amit II-429
Diemert, Denis II-460
Dong, Changyu II-143

E

Eisenbarth, Thomas II-460
Ernst, Johannes II-167
Esser, Andre II-276

F

Faust, Sebastian II-675
Friji, Hamdi I-532
Friolo, Daniele II-307
Frymann, Nick I-394
Fu, Chaohao I-583
Fukushima, Kazuhide I-695
Fyrbiak, Marc II-62

G

Gardham, Daniel I-394
Gérard, Benoît I-725
Ghosh, Shibam I-89
Gigerl, Barbara I-3
Grover, Charlie II-429
Gruss, Daniel I-33
Güneysu, Tim II-62

H

Harmon, Luke I-425
Hazay, Carmit II-611
Hiwatashi, Keitaro II-541
Hou, Xintian II-115
Housni, Youssef El I-339
Hsiao, Hsu-Chun I-260

Hsiao, Shou-Ching I-260
Hua, Jingyu I-476

I

Ikram, Muhammad II-699

J

Jager, Tibor II-460
Jaiswal, Shashwat I-208
Ji, Yunfeng I-607
Jia, Hanyu II-115

K

Kaafar, Mohamed Ali I-284, II-699
Kamphuis, Fredrik II-675
Karati, Sabyasachi I-363
Ke, Pengzhen II-197
Kern, Dustin II-85
Kim, Dan Dongseong I-555
Kim, Tiffany Hyun-Jin I-260
Kiyomoto, Shinsaku I-695
Koutsos, Vlasis II-396
Kraft, Erik I-33
Krauß, Christoph II-85
Kundu, Anup Kumar I-89
Kunihiro, Noboru I-59
Kunzweiler, Sabrina II-276

L

Lagodzinski, J. A. Gregor I-451
Lai, Yi-Fu II-248
Lamberty, Ricky II-675
Lauser, Timm II-85
Levin, Shai II-248
Li, Qiang II-115
Li, Xiangxue II-115
Lim, Hyuk I-555
Lin, Yu I-476
Liu, Yong I-119
Loo, Jia Yi I-503
Low, Ann Tene I-260

M

Ma, Jack P. K. II-484
Magri, Bernardo II-675
Mangard, Stefan I-3
Manulis, Mark I-394
Mao, Yunlong I-476
May, Alexander II-276

Méaux, Pierrick I-634
Meier, Willi I-178
Mitrokotsa, Aikaterini II-167
Moore, Terrence J. I-555

N

Nartz, Hugo I-394
Nelson, Frederica I-555
Ngo, Mao V. I-503
Nguyen, Ky II-336
Niederhagen, Ruben II-85
Niesler, Christian II-32
Nuida, Koji II-541

O

Okada, Hiroki I-695
Olivereau, Alexis I-532

P

Pajola, Luca I-309
Papadopoulos, Dimitrios II-396
Parra-Arnau, Javier I-451
Pébereau, Pierre I-634
Pedersen, Robi II-219
Phan, Duong Hieu II-336
Pointcheval, David II-336, II-516
Primas, Robert I-3

R

Rabadi, Dima I-503
Rahman, Mostafizar I-89
Richmond, Tania I-725
Ricosset, Thomas II-516
Rossi, Mélissa II-516
Roy, Arnab I-425
Ruan, Na I-583

S

Sadeghi, Ahmad-Reza II-32
Sageloli, Éric I-634
Saha, Dhiman I-89, I-208
Salvino, Matteo II-307
Sarkiss, Mireille I-532
Sasdrich, Pascal II-62
Schwarzl, Martin I-33
Sentana, I. Wayan Budi II-699
Shlomovits, Omer II-645
Silva, David I-425
Stolz, Florian II-62

Strufe, Thorsten I-451
Surminski, Sebastian II-32
Suryawanshi, Sahiba I-208

T

Takagi, Tsuyoshi I-695
Tani, Kenta I-59
Tao, Yang I-607
Teh, Je Sen I-149
Teo, Sin G. I-503
Tian, Cong I-237
Tran, Bénédict I-664
Tricomi, Pier Paolo I-309
Truong-Huu, Tram I-503

U

Udovenko, Aleksei I-149

V

Varlakov, Denis II-645
Vaudenay, Serge I-664
Velichkov, Vesselin I-149
Venturi, Daniele II-307

W

Wang, Ping-Lun I-260
Wen, Ting II-582
Weng, Jian II-143
Weninger, Andreas II-3
Wiesmaier, Alexander II-85

Wood, Ian I-284
Wu, Huangting II-366

X

Xia, Yu II-555
Xiang, Zejun I-119
Xie, Hao II-582
Xu, Yongfeng II-115
Xue, Quan II-582

Y

Yanai, Avishay II-611
Yang, Yaxi II-143
Ye, Dengpan I-237
Yi, Yufeng II-143
Yuen, Tsz Hon II-484

Z

Zeng, Xiangyong I-119
Zhang, Leo Yu II-143
Zhang, Liang Feng II-197
Zhang, Qisheng I-555
Zhang, Rui I-607
Zhang, Shasha I-119
Zhang, Tianling I-476
Zhang, Yuan I-476
Zhao, Benjamin Zi Hao I-284
Zhong, Sheng I-476
Zhou, Jianying II-143