

Stephan Krenn
Haya Shulman
Serge Vaudenay (Eds.)

LNCS 12579

Cryptology and Network Security

19th International Conference, CANS 2020
Vienna, Austria, December 14–16, 2020
Proceedings

 Springer

Founding Editors

Gerhard Goos

Karlsruhe Institute of Technology, Karlsruhe, Germany

Juris Hartmanis

Cornell University, Ithaca, NY, USA

Editorial Board Members

Elisa Bertino

Purdue University, West Lafayette, IN, USA

Wen Gao

Peking University, Beijing, China

Bernhard Steffen 

TU Dortmund University, Dortmund, Germany

Gerhard Woeginger 

RWTH Aachen, Aachen, Germany

Moti Yung

Columbia University, New York, NY, USA

More information about this subseries at <http://www.springer.com/series/7410>

Stephan Krenn · Haya Shulman ·
Serge Vaudenay (Eds.)

Cryptology and Network Security

19th International Conference, CANS 2020
Vienna, Austria, December 14–16, 2020
Proceedings

Editors

Stephan Krenn
AIT Austrian Institute of Technology GmbH
Vienna, Austria

Haya Shulman
Fraunhofer SIT
Darmstadt, Germany

Serge Vaudenay
IC LASEC
EPFL
Lausanne, Switzerland

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-030-65410-8 ISBN 978-3-030-65411-5 (eBook)
<https://doi.org/10.1007/978-3-030-65411-5>

LNCS Sublibrary: SL4 – Security and Cryptology

© Springer Nature Switzerland AG 2020

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

The 19th International Conference on Cryptology and Network Security (CANS 2020) was held during December 14–16, 2020, as an online conference, due to the COVID-19 pandemic. CANS 2020 was held in cooperation with the International Association for Cryptologic Research (IACR).

CANS is a recognized annual conference focusing on cryptology, computer and network security, and data security and privacy, attracting cutting-edge research findings from scientists around the world. Previous editions of CANS were held in Taipei ('01), San Francisco ('02), Miami ('03), Xiamen ('05), Suzhou ('06), Singapore ('07), Hong Kong ('08), Kanazawa ('09), Kuala Lumpur ('10), Sanya ('11), Darmstadt ('12), Parary ('13), Crete ('14), Marrakesh ('15), Milan ('16), Hong Kong ('17), Naples ('18), and Fuzhou ('19).

In 2020, the conference received 118 submissions. The submission and review process was done using the EasyChair Web-based software system. We were helped by 40 Program Committee members and 110 external reviewers. The submissions went through a doubly-anonymous review process and 30 papers were selected. This volume represents the revised version of the accepted papers.

Following the CANS tradition, the Program Committee awarded some authors. This year, the Best Paper Award was given for three papers:

- Daniel Kales and Greg Zaverucha for “An Attack on Some Signature Schemes Constructed From Five-Pass Identification Schemes”
- Andrea Caforio, Fatih Balli, and Subhadeep Banik for “Energy Analysis of Lightweight AEAD Circuits”
- Bar Meyuhas, Nethanel Gelernter, and Amir Herzberg for “Cross-Site Search Attacks: Unauthorized Queries over Private Data”

We were honored to have four keynote speakers: Atsuko Miyaji, Kenny Paterson, Mathias Payer, and Zhiyun Qian. We also had a tutorial by Amir Herzberg.

We would like to thank the ATHENE National Research Center for Applied Cybersecurity, as well as the H2020 initiative CyberSec4Europe, for their support during the planning of the conference. We would also like to thank Springer for their support with producing the proceedings. We heartily thank the authors of all submitted papers. Moreover, we are grateful to the members of the Program Committee and the external sub-reviewers for their diligent work, as well as all members of the Organizing Committee for their kind help. We would also like to acknowledge the Steering Committee for supporting us.

November 2020

Stephan Krenn
Haya Shulman
Serge Vaudenay

Organization

Steering Committee

Yvo G. Desmedt (Chair)	The University of Texas at Dallas, USA
Juan A. Garay	Texas A&M University, USA
Amir Herzberg	Bar-Ilan University, Israel
Yi Mu	Fujian Normal University, China
Panos Papadimitratos	KTH, Sweden
David Pointcheval	CNRS, ENS Paris, France
Huaxiong Wang	Nanyang Technological University, Singapore

PC Chairs

Haya Shulman	Fraunhofer SIT, Germany
Serge Vaudenay	Ecole Polytechnique Fédérale de Lausanne, Switzerland

General Chair

Stephan Krenn	AIT Austrian Institute of Technology, Austria
---------------	---

Organizing Committee

Manuela Kos	AIT Austrian Institute of Technology, Austria
Michael Mürling	AIT Austrian Institute of Technology, Austria
Krzysztof Pietrzak (Publicity Chair)	IST Austria, Austria
Hervais Simo	Fraunhofer SIT, Germany

Program Committee

Yehuda Afek	Tel-Aviv University, Israel
Steven Arzt	Fraunhofer, Germany
Xavier Boyen	Queensland University of Technology, Australia
Bremner Bremner-Barr	IDC, Israel
Sherman S. M. Chow	The Chinese University of Hong Kong, Hong Kong
Ran Cohen	Northeastern University, USA
Sabrina De Capitani di Vimercati	Università degli Studi di Milano, Italy
F. Betül Durak	Robert Bosch LLC, USA
Michael Franz	University of California, Irvine, USA
Flavio D. Garcia	University of Birmingham, UK

Peter Gaži	IOHK Research, Slovakia
Niv Gilboa	Ben-Gurion University, Israel
Dieter Gollmann	Hamburg University of Technology, Germany
Louis Goubin	Versailles Saint-Quentin-en-Yvelines University, France
Amir Herzberg	Department of Computer Science and Engineering, Israel
Sotiris Ioannidis	Technical University of Crete, Greece
Alptekin Küpçü	Koç University, Turkey
Atefeh Mashatan	Ryerson University, Canada
Kazuhiko Minematsu	NEC Corporation, Japan
Chris Mitchell	Royal Holloway, UK
Max Mühlhäuser	TU Darmstadt, Germany
Mridul Nandi	Indian Statistical Institute, India
Raphael C.-W. Phan	Monash University, Malaysia
Thomas Pornin	NCC Group, Canada
Neta Rozen-Schiff	Hebrew University of Jerusalem, Israel
Mark Ryan	University of Birmingham, UK
Peter Y. A. Ryan	University of Luxembourg, Luxembourg
Simona Samardjiska	Radboud University, The Netherlands
Gil Segev	Hebrew University of Jerusalem, Israel
Jean-Pierre Seifert	Technical University of Berlin, Germany
Haya Shulman	Fraunhofer, Germany
Cristian-Alexandru Staicu	CISPA Helmholtz Center for Information Security, Germany
Nikhil Tripathi	TU Darmstadt, Germany
Serge Vaudenay	Ecole Polytechnique Fédérale de Lausanne, Switzerland
Damien Vergnaud	Sorbonne Université, Institut Universitaire de France, France
Ivan Visconti	Università degli Studi di Salerno, Italy
Damian Vizár	CSEM, Switzerland
Edgar Weippl	University of Vienna, Austria
Matthias Wählisch	Freie Universität Berlin, Germany
Avishay Yanai	VMware Research, Israel

External Reviewers

Abdulla Aldoseri	Augustin Bariant
Nikolaos Alexopoulos	Khashayar Barooti
Gennaro Avitabile	Andrea Basso
Shahar Azulay	Rishabh Bhaduria
Fatih Balli	Rishiraj Bhattacharyya
Subhadeep Banik	Osman Biçer

Leon Böck
Vincenzo Botta
Andrea Caforio
Fabio Campos
Avik Chakraborti
Gwangbae Choi
Daniel Collins
Sandro Coretti
Hila Dahari
Nilanjan Datta
Mohammad Sadeq Dousti
Cansu Döğanyay
Alexandre Duc
Sonia Duc
Minxin Du
Ehsan Ebrahimi
Rolf Egert
Keita Emura
Daniel Fentham
Georgios Fotiadis
Daniele Friolo
Sarah Gaballah
Reza Ghasemi
Anirban Ghatak
Simin Ghesmati
Satrajit Ghosh
Tim Grube
Mathias Gusenbauer
Ariel Hamlin
Yahya Hassanzadeh-Nazarabadi
Philipp Holzinger
Lois Huguenin-Dumittan
Vincenzo Iovino
Novak Kaluderovic
Shankar Karuppayah
Handan Kılınç Alper
Maria Kober
Philip Kolvenbach
Veronika Kuchta
Mario Larangeira
Eysa Lee
Wanpeng Li
Fukang Liu
Jack P. K. Ma
Alexandra Mai
Lukas Malina
Karola Marky
Adrian Marotzke
Zdenek Martinasek
Soundes Marzougui
Marc Miltenberger
Jose Moreira
Johannes Mueller
Sayantan Mukherjee
Kit Murdock
Alon Noy (Neuhaus)
Lucien K. L. Ng
Mihai Ordean
Shinjo Park
Guillermo Pascual-Perez
Francesco Pasquale
Leo Perrin
Katharina Pfeffer
Niklas Pirnay
Andreea-Ina Radu
Eyal Ronen
Yann Rotella
Lior Rotem
Sujoy Sinha Roy
Pratik Sarkar
Liron Schiff
Philipp Schindler
Gili Schul-Ganz
Henning Seidler
Kris Shrishak
Rajiv Ranjan Singh
Marjan Skrobot
Najmeh Soroush
Aikaterini Sotiraki
Sanaz Taheri-Boshrooyeh
Hiroto Tamiya
Sam L. Thomas
Bénédict Tran
Itay Tsabary
Andrea Tundis
Rei Ueno
Bogdan Ursu
Jeroen van Wier

Benoit Viguiier
Aidmar Wainakh
Jiafan Wang
Thom Wiggers
Nils Wisiol

Donald P. H. Wong
Harry W. H. Wong
Zohar Yakhini
Hailun Yan
Yongjun Zhao

Contents

Best Papers

An Attack on Some Signature Schemes Constructed from Five-Pass Identification Schemes	3
<i>Daniel Kales and Greg Zaverucha</i>	
Energy Analysis of Lightweight AEAD Circuits	23
<i>Andrea Caforio, Fatih Balli, and Subhadeep Banik</i>	
Cross-Site Search Attacks: Unauthorized Queries over Private Data.	43
<i>Bar Meyuhas, Nethanel Gelernter, and Amir Herzberg</i>	

Cybersecurity

Stronger Targeted Poisoning Attacks Against Malware Detection	65
<i>Shintaro Narisada, Shoichiro Sasaki, Seira Hidano, Toshihiro Uchibayashi, Takuo Suganuma, Masahiro Hiji, and Shinsaku Kiyomoto</i>	
STDNeut: Neutralizing Sensor, Telephony System and Device State Information on Emulated Android Environments.	85
<i>Saurabh Kumar, Debadatta Mishra, Biswabandan Panda, and Sandeep K. Shukla</i>	
HMAC and “Secure Preferences”: Revisiting Chromium-Based Browsers Security	107
<i>Pablo Picazo-Sanchez, Gerardo Schneider, and Andrei Sabelfeld</i>	
Detecting Word Based DGA Domains Using Ensemble Models	127
<i>P. V. Sai Charan, Sandeep K. Shukla, and P. Mohan Anand</i>	

Credentials

Distance-Bounding, Privacy-Preserving Attribute-Based Credentials.	147
<i>Daniel Bosk, Simon Bouget, and Sonja Buchegger</i>	
Trenchcoat: Human-Computable Hashing Algorithms for Password Generation	167
<i>Ruthu Hulikal Rooparaghunath, T. S. Harikrishnan, and Debayan Gupta</i>	
Provably Secure Scalable Distributed Authentication for Clouds	188
<i>Andrea Huszti and Norbert Oláh</i>	

Forward-Secure 0-RTT Goes Live: Implementation and Performance Analysis in QUIC 211
Fynn Dallmeier, Jan P. Drees, Kai Gellert, Tobias Handirk, Tibor Jager, Jonas Klauke, Simon Nachtigall, Timo Renzelmann, and Rudi Wolf

Elliptic Curves

Semi-commutative Masking: A Framework for Isogeny-Based Protocols, with an Application to Fully Secure Two-Round Isogeny-Based OT 235
Cyprien Delpèch de Saint Guilhem, Emmanuela Orsini, Christophe Petit, and Nigel P. Smart

Optimized and Secure Pairing-Friendly Elliptic Curves Suitable for One Layer Proof Composition 259
Youssef El Housni and Aurore Guillevic

Curves with Fast Computations in the First Pairing Group 280
Rémi Clarisse, Sylvain Duquesne, and Olivier Sanders

Revisiting ECM on GPUs 299
Jonas Wloka, Jan Richter-Brockmann, Colin Stahlke, Thorsten Kleinjung, Christine Priplata, and Tim Güneysu

Payment Systems

Arcula: A Secure Hierarchical Deterministic Wallet for Multi-asset Blockchains 323
Adriano Di Luzio, Danilo Francati, and Giuseppe Ateniese

Detecting Covert Cryptomining Using HPC 344
Ankit Gangwal, Samuele Giuliano Piazzetta, Gianluca Lain, and Mauro Conti

Lightweight Virtual Payment Channels 365
Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka

Privacy-Enhancing Tools

Chosen-Ciphertext Secure Multi-identity and Multi-attribute Pure FHE 387
Tapas Pal and Ratna Dutta

Linear Complexity Private Set Intersection for Secure Two-Party Protocols 409
Ferhat Karakoç and Alptekin Küpçü

Compact Multi-Party Confidential Transactions. 430
Jayamine Alupotha, Xavier Boyen, and Ernest Foo

Simulation Extractable Versions of Groth’s zk-SNARK Revisited 453
Karim Baghery, Zaira Pindado, and Carla Ràfols

Efficient Composable Oblivious Transfer from CDH in the Global Random Oracle Model 462
Bernardo David and Rafael Dowsley

Lightweight Cryptography

Integral Cryptanalysis of Reduced-Round Tweakable TWINE. 485
Muhammad ElSheikh and Amr M. Youssef

RiCaSi: Rigorous Cache Side Channel Mitigation via Selective Circuit Compilation. 505
Heiko Mantel, Lukas Scheidel, Thomas Schneider, Alexandra Weber, Christian Weinert, and Tim Weißmantel

Assembly or Optimized C for Lightweight Cryptography on RISC-V?. 526
Fabio Campos, Lars Jellema, Mauk Lemmen, Lars Müller, Amber Sprenkels, and Benoit Viguier

Codes and Lattices

Attack on LAC Key Exchange in Misuse Situation 549
Aurélien Greuet, Simon Montoya, and Guénaél Renault

Enhancing Code Based Zero-Knowledge Proofs Using Rank Metric 570
Emanuele Bellini, Philippe Gaborit, Alexandros Hasikos, and Victor Mateu

A Secure Algorithm for Rounded Gaussian Sampling 593
Séamus Brannigan, Maire O’Neill, Ayesha Khalid, and Ciara Rafferty

Accelerating Lattice Based Proxy Re-encryption Schemes on GPUs 613
Gyana Sahu and Kurt Rohloff

Author Index 633

Best Papers



An Attack on Some Signature Schemes Constructed from Five-Pass Identification Schemes

Daniel Kales^{1(✉)} and Greg Zaverucha²

¹ Graz University of Technology, Graz, Austria
daniel.kales@iaik.tugraz.at

² Microsoft Research, Redmond, WA, USA
gregz@microsoft.com

Abstract. We present a generic forgery attack on signature schemes constructed from 5-round identification schemes made non-interactive with the Fiat-Shamir transform. The attack applies to ID schemes that use parallel repetition to decrease the soundness error. The attack can be mitigated by increasing the number of parallel repetitions, and our analysis of the attack facilitates parameter selection.

We apply the attack to MQDSS, a post-quantum signature scheme relying on the hardness of the MQ-problem. Concretely, forging a signature for the L1 instance of MQDSS, which should provide 128 bits of security, can be done in $\approx 2^{95}$ operations. We verify the validity of the attack by implementing it for round-reduced versions of MQDSS, and the designers have revised their parameter choices accordingly.

We also survey other post-quantum signature algorithms and find the attack succeeds against PKP-DSS (a signature scheme based on the hardness of the permuted kernel problem) and list other schemes that may be affected. Finally, we use our analysis to choose parameters and investigate the performance of a 5-round variant of the Picnic scheme.

Keywords: Public-key signatures · Security analysis · Post-quantum cryptography · Fiat-Shamir transform · MQDSS

1 Introduction

Digital signatures are one of the fundamental cryptographic building blocks and are widely used for authentication of data and in protocols. Recently, advances in quantum computing have motivated new designs for digital signature schemes, having post-quantum security, i.e., schemes that are implemented on classical computers but have security against attacks by quantum computers. NIST has started a standardization project for post-quantum cryptographic primitives [24].

A popular approach to designing signature schemes is to start with an interactive identification (ID) scheme and use the Fiat-Shamir transformation to make

D. Kales—Part of this work was conducted during an internship at Microsoft Research.

© Springer Nature Switzerland AG 2020

S. Krenn et al. (Eds.): CANS 2020, LNCS 12579, pp. 3–22, 2020.

https://doi.org/10.1007/978-3-030-65411-5_1

it non-interactive and transform it into a signature scheme. The most well-known example of this is Schnorr’s ID and signature scheme. Multiple signature schemes with conjectured post-quantum security also use this approach (e.g., Dilithium, MQDSS, Picnic, PKP-DSS, and qTESLA, among others).

However, while the Fiat-Shamir transform for 3-round (also called 3-pass) ID schemes (where a total of three messages are exchanged between the prover and verifier) is well understood, some signatures are built on identification schemes with five or more rounds. One such example is MQDSS, which builds on the 5-pass identification scheme of Sakumoto et al. [27]. Chen et al. [12] give a construction for a Fiat-Shamir transformation for a certain class of 5-pass identification schemes, which includes the one from [27], and use it to build MQDSS.

ID schemes are closely related to zero-knowledge proofs and arguments, and some of the terminology is shared; we often refer to the parties as prover and verifier, the prover’s secret is called a witness, and the public key is called a statement. The soundness error of a proof protocol, denoted ϵ , is the probability that a malicious prover can get a verifier to accept without knowing the witness. For κ -bit security, we thus require $\epsilon < 2^{-\kappa}$.

Some proof protocols have a large constant soundness error, such as $\epsilon = 1/2$. In this case, we can hope to amplify the soundness of the protocol by repeating the protocol r times. In the best case, the effect is exponential, and r repetitions give soundness error of ϵ^r . This is known to be the case for interactive protocols when the repetitions are performed sequentially. The question for parallel repetition has been a topic of study for many years.

Parallel repetition does decrease soundness error exponentially for 3-round, public-coin protocols (i.e., protocols where the verifier has no secret key) [4]. For more than three rounds there are examples of non-public coin protocols where parallel repetition is not effective [25], but when considering only public-coin protocols parallel repetition is effective [18].

However, these positive results only apply to interactive protocols, and only hold asymptotically, so they do not give a concrete number of repetitions for κ bits of security. Intuitively, soundness for non-interactive protocols can only be worse, since the verifier’s steps can be implemented by a malicious prover and run many times during a search for a cheating proof. Thus, choosing r to achieve κ -bit security for concrete non-interactive proof protocols and signature schemes is not obvious, especially so for protocols with more than three rounds. Choosing r such that $\epsilon^r < 2^{-\kappa}$ seems to work for three-round protocols (we are not aware of cases where this fails), so we name this approach the ϵ^r -heuristic. As we will show, the ϵ^r -heuristic does not hold for five-round protocols. Instead, the secure choice of r is a function of ϵ and the challenge spaces of the protocol.

Contributions. In this paper, we give a generic attack on five-round identification schemes made non-interactive using the Fiat-Shamir transform. This shows that the ϵ^r -heuristic fails for non-interactive 5-round protocols, and care must be taken when choosing r . The concrete attack complexity is influenced only by the size of the challenge spaces in the different rounds and whether the identification scheme has a property we call *capability for early abort*. We give general formulas

for the attack complexity and show how this influences the parameter choices of different signature schemes, and discuss strategies for designers using 5-round protocols to build signatures.

As an application of our result, we show an attack on the proposed parameter sets for MQDSSv2. We show that at the 128-bit security level, our attack finds forgeries with 2^{95} operations. We practically verify the attack on round-reduced versions of MQDSS and discuss ways to reduce its practical complexity. The designers of MQDSS have confirmed our attack and changed their proposed instances according to our recommendations during the latest round of updates in the NIST post-quantum standardization project (MQDSSv2.1).

Even though the construction of [12] has a security proof, its non-tightness allows for the attack to exist, i.e., the attack does not contradict the asymptotic security reduction, and takes exponential time. This is an example of a non-tight proof reflecting the real-world security a scheme. This is somewhat rare, and has been called the “nightmare scenario” by Menezes [23, §5.4] since there are many examples of non-tight proofs where security is thought to hold for the natural choice of parameters (Schnorr signatures being a prominent example). To our knowledge this is the first such example for a public-key signature scheme.

Our attack also applies to PKP-DSS [8], a signature scheme based on a five-round proof protocol for the permuted kernel problem [28]. The latest proposed parameters for PKP-DSS have also been updated to account for our attack.

We also use our analysis to select parameters for a 5-round version of the Picnic signature scheme [11, 31], and quantify the resulting performance. The designers chose to collapse the 5-round protocol to three rounds with only an informal justification, and left open the question of using five rounds. Our analysis confirms that the parameters of the underlying proof system [21] would need to increase such that both the proof size and runtime is better when using the three-round variant.

1.1 Additional Related Work

In [22], Kiltz et al. give a tight security proof for signatures based on a Fiat-Shamir transformation of a 5-pass identification scheme. For their proof, they require the underlying identification scheme to have a property called security against non-adaptive parallel impersonation key-only attacks (naPIMP-KOA). However, they do not consider the case of parallel repetition, which is needed if the soundness error of a single invocation of the identification scheme is not small enough. Although their FS transformation has slight differences to the one used in MQDSS, these differences are minor, and our attack can also be adapted to the transformation of [22] in case parallel repetition is used.

Five-to-Three Round Signature Schemes. The Picnic signature scheme instances based on the KKW proof system [21], have a 5-round structure, arising from the choice of MPC protocol used to implement the MPC-in-the-head construction [19]. The protocol has a preprocessing phase used to establish correlated randomness between the parties, to be used in an online phase. In the KKW proof

protocol, the first and second rounds correspond to the preprocessing and online phases (resp.). By performing the online phase for all preprocessing instances, [21] carefully collapses the 5-round protocol to three rounds.

In [6], Beullens generalizes KKW to design sigma protocols for the permuted kernel problem (PKP), solutions of multivariate quadratic (MQ) equation systems and the shortest integer solution (SIS) in lattices. These sigma protocols are named “sigma protocols with helper”, where a trusted third party acts as a helper to set up correlated randomness in a preprocessing step. The trusted third party is then replaced by a cut-and-choose approach as in [21]. The overall structure of these sigma protocols is also 5-round, collapsed to three.

These five-to-three schemes then *beat* the ϵ^r -heuristic, by doing the cut-and-choose step across all parallel repetitions, effectively replacing the independent parallel repetitions with a single repetition.

In [3] Baum and Nof give interactive five-round protocols for proving knowledge of a solution to the short integer solution lattice problem. Their work also generalizes [21] but changes the cut-and-choose step to use the sacrificing technique. A direct application of the Fiat-Shamir transform to these protocols would need to address our attack.

There are many 5-pass identification protocols for code and lattice problems that are inspired by early three and 5-pass protocols based on syndrome decoding by Stern [29,30]. Stern uses the ϵ^r -heuristic when discussing the signature scheme associated with the 3-pass variant of his scheme, then presents the 5-pass variant without re-visiting the choice of r . In [9] Cayrel et al. present a lattice-based threshold ring signature scheme based on 5-round identification schemes with soundness error $1/2$ and $1/3$, and choose the number of parallel repetitions using the ϵ^r -heuristic. Some follow-up papers in this area [5,14] also use the ϵ^r -heuristic.

In [10] Cayrel et al. describe an interactive 5-pass identification scheme based on the q -ary syndrome decoding problem with $\epsilon = 1/2$. Then in [16], El Yousfi Alaoui et al. rely on previous analysis of the Fiat-Shamir transform for 5-pass schemes [2] that does not consider parallel repetition in detail and use the ϵ^r -heuristic to choose parameters and benchmark the resulting signature scheme. Similarly, Aguilar et al. [1] present a new five-pass ID scheme based on the syndrome decoding, and choose parameters for the associated signature scheme using the ϵ^r -heuristic, and these parameters were used in the implementation and performance comparison of Dambra et al. [15]. We have not done a thorough analysis of these code-based signatures to conclude which are impacted by our attack, and by how much.

2 Preliminaries

We give a short background on generic 5-pass ID schemes, and the MQDSS signature scheme. We keep the notation consistent with the MQDSS specification document [13], and denote by $\overset{\$}{\leftarrow}$ the uniform random sampling from a set.

2.1 Canonical $(2n + 1)$ -Pass Identification Schemes

Canonical $(2n + 1)$ -pass identification schemes are a class of ID schemes which follow a certain message structure. First, the prover sends an initial commitment com , then the two parties engage in n rounds, where the verifier sends a challenge ch_i drawn from the corresponding challenge set ChS_i , to which the prover responds with rsp_i . We depict a 5-pass identification scheme in Fig. 1. Such identification schemes can be made non-interactive using the Fiat-Shamir transformation [17], replacing the job of the verifier by calls to random functions, usually instantiated using cryptographic hash functions. We give details of this process in the caption of Fig. 1.

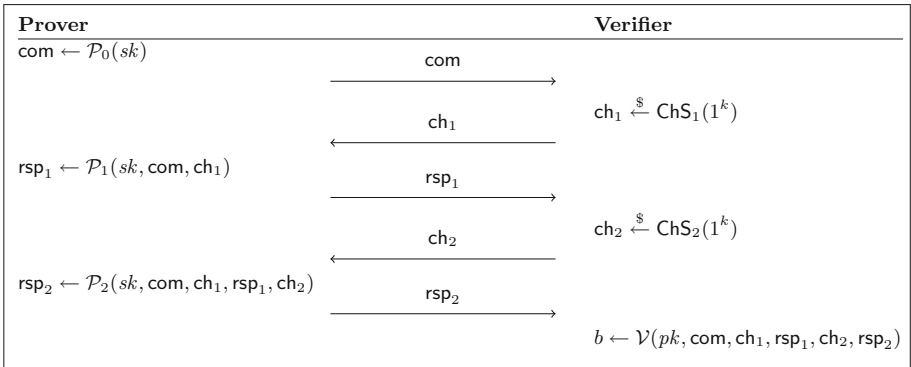


Fig. 1. A canonical 5-pass identification scheme. To make the scheme non-interactive, ch_1 and ch_2 are computed as $\text{ch}_1 = \mathcal{H}_1(\text{com})$ and $\text{ch}_2 = \mathcal{H}_2(\text{com}, \text{ch}_1, \text{rsp}_1)$ for cryptographic hash functions \mathcal{H}_1 and \mathcal{H}_2 . The proof is $\pi = (\text{com}, \text{rsp}_1, \text{rsp}_2)$. In a signature scheme, the message m is included in both H_1 and H_2 and π is the signature on m .

2.2 Fiat-Shamir Transformation for a Class of 5-Pass ID Schemes

In [12], Chen et al. give a Fiat-Shamir transformation for a certain class of 5-pass identification schemes. They note that many existing 5-pass identification schemes follow a certain structure, which they call a $q2$ -identification scheme ($q2$ -IDS), where the challenge spaces have sizes q and 2, respectively. The class of $q2$ -ID schemes may have an associated $q2$ -extractor, which extracts a witness from two accepting but different transcripts. See the full version of this work [20] for a precise definition.

The soundness error of an identification scheme, denoted ϵ , is the probability that the $q2$ -extractor fails. The soundness error can be boosted by running r parallel repetitions of the scheme. Chen et al. [12] use a variant of the Fiat-Shamir transformation in Fig. 1 to turn a $q2$ -IDS into a signature scheme and provide analysis and a security proof for their Construction 1 in the random oracle model (ROM), if the $q2$ -IDS additionally has a $q2$ -extractor.

Construction 1 (Fiat-Shamir transform for $q2$ -IDS [12]). Let $\kappa \in \mathbb{N}$ be the security parameter, $\text{IDS} = (\text{KGen}, \mathcal{P}, \mathcal{V})$ a $q2$ -Identification scheme that achieves soundness with constant soundness error ϵ . Select r the number of (parallel) repetitions of IDS , such that $\epsilon^r = \text{negl}(\kappa)$, and that the challenge spaces of the composition $\text{IDS}^r, C_1^r, C_2^r$ have size exponential in κ . Moreover, select cryptographic hash functions $H_1 : \{0, 1\}^* \mapsto C_1^r$ and $H_2 : \{0, 1\}^* \mapsto C_2^r$. The $q2$ -signature scheme $q2\text{-Dss}(1^\kappa)$ derived from IDS is the triple of algorithms $(\text{KGen}, \text{Sign}, \text{Vf})$ with:

- $(sk, pk) \leftarrow \text{KGen}(1^\kappa)$
- $\sigma = (\sigma_0, \sigma_1, \sigma_2) \leftarrow \text{Sign}(sk, m)$ where $\sigma_0 = \text{com} \leftarrow \mathcal{P}_0^r(sk), h_1 = H_1(m, \sigma_0), \sigma_1 = \text{rsp}_1 \leftarrow \mathcal{P}_1^r(sk, \sigma_0, h_1), h_2 = H_2(m, \sigma_0, h_1, \sigma_1)$ and $\sigma_2 = \text{rsp}_2 \leftarrow \mathcal{P}_2^r(sk, \sigma_0, h_1, \sigma_1, h_2)$.
- $\text{Vf}(pk, m, \sigma)$ parses $\sigma = (\sigma_0, \sigma_1, \sigma_2)$, computes the values $h_1 = H_1(m, \sigma_0), h_2 = H_2(m, \sigma_0, h_1, \sigma_1)$ as above and outputs $\mathcal{V}^r(pk, \sigma_0, h_1, \sigma_1, h_2, \sigma_2)$.

Theorem 1 (EU-CMA security of $q2$ -signature schemes [12]). Let $\kappa \in \mathbb{N}$, $\text{IDS}(1^\kappa)$ be a $q2$ -IDS that is honest-verifier zero-knowledge, achieves soundness with constant soundness error ϵ and has a $q2$ -extractor. Then $q2\text{-Dss}(1^\kappa)$, the $q2$ -signature scheme derived applying Construction 1 is existentially unforgeable under adaptive chosen message attacks.

The proof of Theorem 1 is given in [12]. However, the authors also note that the proof is non-tight due to its use of the forking lemma [26]. The number of parallel repetitions r are chosen according to the ϵ^r -heuristic, based the soundness error of the underlying IDS , ignoring the potential loss in security that comes from the non-tightness of the proof.

3 Forgery Attacks on MQDSS

Chen et al. [12] give a concrete instantiation – called MQDSS – by applying Construction 1 to the 5-pass identification scheme from Sakumoto et al. [27]. MQDSS is a post-quantum signature scheme submitted to the NIST post-quantum standardization project. We first recall the details of the MQDSS signature scheme and then describe our attack.

3.1 Description of MQDSS

The main idea of the 5-pass identification scheme by Sakumoto et al. [27] is to prove knowledge of a solution \mathbf{s} of a multivariate quadratic equation system $\mathbf{v} = \mathbf{F}(\mathbf{s})$. To achieve this, the secret \mathbf{s} is split into two shares $\mathbf{s} = \mathbf{r}_0 + \mathbf{r}_1$ and the public key \mathbf{v} can be represented using the polar form of \mathbf{F} as $\mathbf{v} = \mathbf{F}(\mathbf{r}_0) + \mathbf{F}(\mathbf{r}_1) + \mathbf{G}(\mathbf{r}_0, \mathbf{r}_1)$. One of the shares of the secret (with an additional masking factor α) is then split further, so that the polar form is not dependent on both shares of the secret: $\alpha\mathbf{r}_0 = \mathbf{t}_0 + \mathbf{t}_1$ and $\alpha\mathbf{F}(\mathbf{r}_0) = \mathbf{e}_0 + \mathbf{e}_1$. Due to the properties of the polar form, we arrive at the relation

$$\alpha\mathbf{v} = (\mathbf{e}_1 + \alpha\mathbf{F}(\mathbf{r}_1 + \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1))) + (\mathbf{e}_0 + \mathbf{G}(\mathbf{t}_0, \mathbf{r}_1)),$$

where each of the two separate summands does not reveal any information about the secret. This is used in the identification protocol where one of these summands is revealed to the verifier and checked for consistency. For more details we refer to [27, Section 4].

The key generation of MQDSS samples a \mathcal{MQ} relation $\mathbf{v} = \mathbf{F}(\mathbf{s})$, but does so pseudorandomly from a k -bit seed sk , by using SHAKE256 as a pseudorandom generator (PRG) and using rejection sampling to sample field elements when necessary. The function $\text{XOF}_{\mathbf{F}}$ generates a multivariate system from a seed, and \mathcal{H} , H_1 , and H_2 are cryptographic hash functions. In the MQDSS signing algorithm the secret key sk is, as in key generation, expanded into four seeds. These seeds are used to derive the \mathcal{MQ} relation and, in combination with a pseudorandom salt D , the shares of the secret \mathbf{r} , \mathbf{t} , \mathbf{e} and the commitment randomness ρ . We can observe the 5-pass structure with the five messages $\sigma_0, \text{ch}_1, \sigma_1, \text{ch}_2$ and σ_2 . For a complete description of the MQDSS key generation, signing and verification, we refer to the MQDSS design document [13] or the full version of this work [20].

MQDSS Versions. In August 2018, the MQDSS team updated their specification and recommended parameter sets, due to the original parameters mistakenly being selected for a higher security level. This new parameter sets were called MQDSS v1.1. Additionally, in March 2019 the MQDSS team modified the scheme to include a random string ρ of length 2κ in their commitments, resulting in MQDSS v2.0. Our attack applies to both, MQDSS v1.1 and v2.0, but in the following, we will use MQDSS v2.0 to be compatible with the most recent reference implementation. After disclosing our attack to the authors, they updated their parameter sets to resist our attack. At the time of writing, v2.1 is the most recent version of MQDSS.

Table 1. Parameter sets for MQDSS instances. r is the number of parallel repetitions in MQDSS v2.0, r_{new} is the number of repetitions required to resist our attack. (Instance for security level L5 not officially submitted to NIST). τ^* is the optimal number of repetitions to attack in the first phase, while $\#\mathcal{H}$ gives an estimate of the required hash function calls for a single forgery.

Parameter Set	κ	$m = n$	q	r	τ^*	$\#\mathcal{H}$	r_{new}
MQDSS-toy	38	48	31	40	11	2^{29}	53
MQDSS-L1	128	48	31	135	41	2^{95}	184
MQDSS-L3	192	64	31	202	61	2^{141}	277
MQDSS-L5	256	88	31	268	82	2^{180}	370

Algorithm 1. Forge(pk, Msg)

```

Parse  $pk$  as  $S_F, \mathbf{v}$ 
 $\mathbf{F} \leftarrow \text{XOF}_F(S_F)$ 
 $\mathbf{r}_0^{(1)}, \dots, \mathbf{r}_0^{(r)}, \mathbf{t}_0^{(1)}, \dots, \mathbf{t}_0^{(r)}, \mathbf{e}_0^{(1)}, \dots, \mathbf{e}_0^{(r)} \xleftarrow{\$} \mathbb{F}_q^{n \times 3r}$ 
 $\alpha^* \xleftarrow{\$} \mathbb{F}_q$ 
 $\mathbf{s}^* \xleftarrow{\$} \mathbb{F}_q^n$ 
for  $j \in \{1, \dots, r\}$  do
   $\mathbf{r}_1^{(j)} \leftarrow \mathbf{s}^* - \mathbf{r}_0^{(j)}$ 
   $\mathbf{t}_1^{(j)} \leftarrow \alpha^* \cdot \mathbf{r}_0^{(j)} - \mathbf{t}_0^{(j)}$ 
   $\mathbf{e}_1^{(j)} \leftarrow \alpha^* \cdot \mathbf{F}(\mathbf{r}_0^{(j)}) - \mathbf{e}_0^{(j)}$ 
   $\rho_0^{(j)}, \rho_1^{(j)} \xleftarrow{\$} \{0, 1\}^{2\kappa \times 2}$ 
   $\text{com}_0^{(j)} \leftarrow \mathcal{H}(\rho_0^{(j)}, \mathbf{r}_0^{(j)}, \mathbf{t}_0^{(j)}, \mathbf{e}_0^{(j)})$ 
   $\text{com}_1^{(j)} \leftarrow \mathcal{H}(\rho_1^{(j)}, \mathbf{r}_1^{(j)}, \alpha^* \cdot (\mathbf{v} - \mathbf{F}(\mathbf{r}_1^{(j)})) - \mathbf{G}(\mathbf{t}_1^{(j)}, \mathbf{r}_1^{(j)}) - \alpha^* \cdot \mathbf{F}(\mathbf{r}_0^{(j)}) + \mathbf{e}_0^{(j)})$ 
end for
 $\sigma_0 \leftarrow \mathcal{H}(\text{com}_0^{(1)}, \text{com}_1^{(1)}, \dots, \text{com}_0^{(r)}, \text{com}_1^{(r)})$ 
repeat
   $R \xleftarrow{\$} \{0, 1\}^{2\kappa}$ 
   $D \leftarrow \mathcal{H}(pk || R || Msg)$ 
   $\text{ch}_1 \leftarrow H_1(D, \sigma_0)$ 
  Parse  $\text{ch}_1$  as  $\text{ch}_1 = \{\alpha^{(1)}, \dots, \alpha^{(r)}\}, \alpha^{(j)} \in \mathbb{F}_q$ 
until at least  $\tau^*$  of  $\alpha^{(j)}$  are equal to  $\alpha^*$ 
repeat
   $\text{guess} \xleftarrow{\$} \{0, 1\}^r$  // in practice, a counter is used to ensure unique hash inputs
  for  $j \in \{1, \dots, r\}$  do
    if  $\alpha^{(j)} = \alpha^*$  then
       $\text{rsp}_1^{(j)} \leftarrow (\mathbf{t}_1^{(j)}, \mathbf{e}_1^{(j)})$ 
    else if bit  $j$  of  $\text{guess}$  is 0 then
       $\text{rsp}_1^{(j)} \leftarrow (\alpha^{(j)} \cdot \mathbf{r}_0^{(j)} - \mathbf{t}_0^{(j)}, \alpha^{(j)} \cdot \mathbf{F}(\mathbf{r}_0^{(j)}) - \mathbf{e}_0^{(j)})$ 
    else
       $\text{rsp}_1^{(j)} \leftarrow (\mathbf{t}_1^{(j)}, (\alpha^{(j)} - \alpha^*) \cdot (\mathbf{v} - \mathbf{F}(\mathbf{r}_1^{(j)})) + \alpha^* \cdot \mathbf{F}(\mathbf{r}_0^{(j)}) - \mathbf{e}_0^{(j)})$ 
    end if
  end for
   $\sigma_1 \leftarrow (\text{rsp}_1^{(1)}, \dots, \text{rsp}_1^{(r)})$ 
   $\text{ch}_2 \leftarrow H_2(D, \sigma_0, \text{ch}_1, \sigma_1)$ 
until bits of  $\text{ch}_2$  agree with  $\text{guess}$  in positions  $j$  where  $\alpha^{(j)} \neq \alpha^*$ 
 $\sigma_2 \leftarrow (\mathbf{r}_{b(1)}^{(1)}, \dots, \mathbf{r}_{b(r)}^{(r)}, \text{com}_{1-b(1)}^{(1)}, \dots, \text{com}_{1-b(r)}^{(r)}, \rho_{b(1)}^{(1)}, \dots, \rho_{b(r)}^{(r)})$ 
return  $\sigma = (R, \sigma_0, \sigma_1, \sigma_2)$ 

```

3.2 Description of the Attack on MQDSS

The basic idea of the attack is to split the attacker work between two phases: we try to guess ch_1 for τ^* repetitions, and then move on to guess ch_2 for the remaining repetitions. For many 5-pass identification schemes, including the one used in MQDSS, guessing just one of the two challenges correctly allows the

prover to cheat. In the non-interactive version, we leverage the fact that these phases can be repeatedly and separately attacked offline.

In [12], Chen et al. give a basic strategy for a cheating adversary, that works as follows: The cheater chooses α^* as guess for ch_1 and uses a randomly chosen secret key \mathbf{s}^* . He follows the protocol as specified, but computes $\mathbf{r}_1 = \mathbf{s}^* - \mathbf{r}_0$, $\mathbf{t}_1 = \alpha^* \mathbf{r}_0 - \mathbf{t}_0$ and $\mathbf{e}_1 \leftarrow \alpha^* \cdot \mathbf{F}(\mathbf{r}_0) - \mathbf{e}_0$ instead. He also computes the commitment $(\text{com}_0, \text{com}_1)$ as $\text{com}_0 \leftarrow \mathcal{H}(\rho_0, \mathbf{r}_0, \mathbf{t}_0, \mathbf{e}_0)$ and $\text{com}_1 \leftarrow \mathcal{H}(\rho_1, \mathbf{r}_1, \alpha^* \cdot (\mathbf{v} - \mathbf{F}(\mathbf{r}_1)) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) - \alpha^* \cdot \mathbf{F}(\mathbf{r}_0) + \mathbf{e}_0)$. If ch_2 is equal to 0, the recomputed check does not involve the public key \mathbf{v} and will therefore always pass. For $\text{ch}_2 = 1$, the cheater set up the values in a way that the check will still pass if ch_1 was equal to α^* . For our attack, it is important that a bad guess for α^* (i.e., ch_1) can be masked by a correct guess of ch_2 , without the verifier noticing. This fact allows us to improve the basic attack strategy by trying to guess ch_1 for all parallel repetitions (and subsequently fixing any bad guesses in phase 2), not only for a predetermined subset of the repetitions, increasing the success probability to guess τ^* first round challenges correctly from $(\frac{1}{q})^{\tau^*}$ to $P_1(\tau^*)$ as given by Eq. 1.

Our cheater now has the problem of how to efficiently generate different inputs (still passing verification) to the challenge hash functions H_1 and H_2 . For phase 1, this is quite easy, since the signature includes a random salt value R , which is allowed to be chosen freely by the attacker. Therefore an attacker can fix a guess of α^* once, compute the first message σ_0 , and then try different values of R until τ^* of the first challenges agree with α^* . For the second phase, we have already fixed R and can therefore not use the same strategy. However, we can modify the values sent in the second message σ_1 in the following way. While the values of \mathbf{t}_1 and \mathbf{e}_1 computed as given by the cheating strategy outlined above are always correct for $\text{ch}_2 = 0$, and fail to verify for $\text{ch}_2 = 1$, we can also come up with different \mathbf{t}_1 and \mathbf{e}_1 that are correct for $\text{ch}_2 = 1$, but fail for $\text{ch}_2 = 0$. To achieve this we use the same \mathbf{t}_1 , but compute \mathbf{e}_1 such that it corrects the error in com_1 , specifically as

$$\mathbf{e}_1 \leftarrow (\alpha^{(j)} - \alpha^*) \cdot (\mathbf{v} - \mathbf{F}(\mathbf{r}_1)) + \alpha^* \cdot \mathbf{F}(\mathbf{r}_0) - \mathbf{e}_0.$$

Now our attacker has two possible values to send in the second phase of each repetition, enabling him to try $2^{r-\tau^*}$ different inputs to H_2 , and with high probability one of those inputs results in the correct guess for all ch_2 for the remaining $r - \tau^*$ repetitions. The full attack is given in Algorithm 1.

Alternative for Phase 2. Instead of the adversary trying all different combinations as shown above, he can also fix all but the last repetition, and just vary the responses for this last repetition in the following way. Choose a random \mathbf{t}_1 and then calculate \mathbf{e}_1 as $\mathbf{e}_1 \leftarrow (\alpha^{(j)} - \alpha^*) \cdot (\mathbf{v} - \mathbf{F}(\mathbf{r}_1)) + \alpha^* \cdot \mathbf{F}(\mathbf{r}_0) + \mathbf{G}(\mathbf{t}_0, \mathbf{r}_1) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) - \mathbf{e}_0$. This response is always valid for $\text{ch}_2 = 1$. Due to choosing \mathbf{t}_1 at random, we have q^n different possible hash inputs. This method allows us to fix all other repetitions but requires us to always calculate $\mathbf{G}(\mathbf{t}_1, \mathbf{r}_1)$ instead of being able to cache the output once as we can do for the other variant. Additionally,

this can be used in combination with the other strategy, especially for the case when we exhaust all $2^{r-\tau^*}$ possible inputs to H_2 , allowing us to continue the attack without having to repeat the first phase.

3.3 Attack Parameters and Mitigation

For the attack, we want to achieve an optimal tradeoff between the work needed for passing the first phase and the work needed for passing the second phase. If we guess τ^* challenges for the first phase correctly, we can answer both possible challenges for these correct guesses in the second phase, only needing to correctly guess the remaining $r - \tau^*$ second round challenges.

The probability of guessing at least τ first-round challenges from a challenge space of size $|C_1| = q$ correctly is given by Eq. 1:

$$P_1(\tau, r, q) = \Pr \left(\begin{array}{c} \text{guess at least } \tau \text{ of} \\ r \text{ challenges with} \\ \text{size } q \end{array} \right) = \sum_{k=\tau}^r \left(\frac{1}{q}\right)^k \left(\frac{q-1}{q}\right)^{r-k} \binom{r}{k}. \quad (1)$$

To achieve the best tradeoff in terms of attack efficiency, we want to minimize the total work for completing both phases. Therefore, the optimal number of repetitions to attack in the first phase is given by

$$\tau^* = \arg \min_{0 \leq \tau \leq r} \left\{ \frac{1}{P_1(\tau, r, q)} + 2^{r-\tau} \right\},$$

assuming that both phases are of equal cost. We give some discussion of the cost of the two phases in Sect. 3.4. A slightly better choice of τ^* might be possible by weighting the cost of each phase, based on the concrete costs of a given attack implementation.

We give an optimal choice for τ^* for different instances of MQDSS in Table 1, together with the estimated number of random oracle calls for a single forgery and the number of parallel repetitions r_{new} that are required so that the expected number of random oracle calls for this attack is at least 2^k . After communicating the attack to the MQDSS designers, they have updated their specification to our recommended number of repetitions in MQDSS version 2.1.

Comparison to a 3-Pass Version of MQDSS. In [27], Sakumoto et al. additionally give a 3-pass variant of their \mathcal{MQ} -based identification scheme. Chen et al. [12] motivated their choice of the 5-pass variant over the 3-pass variant by the lower resulting signature size of the 5-pass variant. However, in light of our new attacks and the resulting increase in parameters to prevent it, this conclusion is no longer as clear as it used to be. In [12, Appendix A], Chen et al. discuss parameters for the 3-pass scheme and come to a signature size of 54.81 KB for the L5 security level. Based on the formulas given in [12, 13] the 3-pass signature size for the L1 security level would be approximately 27.7 KB, whereas the signature size for the updated parameters of the 5-pass signature is now 27.73 KB, almost exactly equal. The impact of our attack therefore arguably makes the 3-pass variant of MQDSS a more natural choice, since 3-pass schemes are more common, and their security is arguably better understood.

3.4 Practical Verification

To verify the validity of the attack, we implemented it and attacked versions of MQDSS with reduced r . The code is based on the reference implementation of MQDSS¹ and is available at <https://github.com/dkales/MQDSS-forgery>. Our MQDSS-toy instance from Table 1 has the same parameters as the instance for the L1 security level, however, we reduced the number of parallel repetitions of the underlying identification scheme from 135 to 40. Since the soundness error of one instance of the identification scheme of [27] is $\epsilon = \frac{1}{2} + \frac{1}{2q}$, these 40 repetitions should provide about 38 bits of security based on the analysis of Construction 1 by Chen et al. The underlying \mathcal{MQ} problem instance is not modified and still provides 128-bit security against attacks on the \mathcal{MQ} problem itself.

Based on our analysis in Sect. 3.3, we choose the number of repetitions to attack in phase 1 to be $\tau^* = 11$. The estimated number of random oracle calls is approximately 2^{29} , while for our experiments the average over 10 runs is $2^{27.98}$, all taking between 1 and 12 min on a standard desktop PC. Additional details of our implementation are given in the full version of this work [20].

4 Attacks on Five Round Protocols Using the Fiat-Shamir Transform

In this section, we generalize the attack described on MQDSS in the previous section to a canonical five-round proof protocol and discuss choosing a secure number of parallel repetitions. We also give guidance to protocol designers to increase the costs of our attack, to reduce the number of parallel repetitions required for a given security level. We end the section with a brief discussion of the more general $(2n + 1)$ -round protocols.

Recall the general structure of a 5-pass identification scheme from Fig. 1. In the attack on MQDSS in Sect. 3.2, we observed that an attacker could mask a bad guess for the first challenge with a correct guess for the second challenge. However, this is not the case in general. Some 5-pass identification schemes have the capability for *early abort*. We formalize this as an additional verification function $\mathcal{V}_{\text{early}}(\text{com}, \text{ch}_1, \text{rsp}_1)$ that enables the verifier to check the validity of the first three messages. Conceptually, this can also be seen as splitting the protocol into two interleaved 3-pass protocols.

Even if such an algorithm $\mathcal{V}_{\text{early}}$ is not specified explicitly for a scheme, i.e., it may be implicitly contained in \mathcal{V} , we are interested in its theoretical existence since it would allow the verifier to detect wrong guesses for ch_1 , affecting the complexity of the attack.

For identification schemes where no such algorithm exists (e.g., MQDSS), we can employ the improved attack strategy of trying to guess all first-round challenges, subsequently fixing bad guesses in the second challenge. However, if $\mathcal{V}_{\text{early}}$ exists, a malicious prover has to select the parallel repetitions to attack beforehand, increasing the complexity of the attack. Protocols that use the first

¹ <https://github.com/joostrijneveld/MQDSS/tree/NIST>.

challenge in a cut-and-choose construction, where the prover commits to a large set of values, and only some of them are revealed to the verifier, usually allow for the existence of such an early verification algorithm. As an example, we will cover the five-round variant of the KKW [21] proof protocol in Sect. 5.1.

Recall that for an identification scheme to be honest-verifier zero-knowledge (HVZK, a requirement for security of the associated Fiat-Shamir signature), there must exist a simulator \mathcal{S} that, given pk , outputs simulated transcripts of protocol executions between \mathcal{P} and \mathcal{V} , which are indistinguishably distributed from real protocol executions. (For a formal definition, see [12, Def. 2.5].)

For our attack to apply to a canonical 5-pass ID scheme, it must satisfy a stronger type of simulation that we call *piecewise simulatability*. Informally, this means that \mathcal{S} can be refactored (in two different ways) to output the transcript in two parts, allowing for one of the challenges to be chosen as an input. In contrast to standard simulators, which output the whole transcript on input pk , piecewise simulators are a more limited class of algorithms. However, since the simulator is always able to choose at least one of the challenges by itself, it can function without knowledge of the secret. Although piecewise simulatability is a stronger assumption, it is fulfilled by all of the schemes we investigate in this work.

Definition 1. *We say that a 5-round, HVZK ID scheme is piecewise simulatable if there exists algorithms (A_1, A_2) and (B_1, B_2) , defined as follows:*

<u>Simulator A</u>	<u>Simulator B</u>
$A_1(pk)$ outputs $T_1 := (\text{com}, \text{ch}_1, \text{rsp}_1)$	$B_1(pk)$ outputs $T_1 := (\text{com}')$
$A_2(pk, T_1, \text{ch}_2^*)$ outputs (rsp_2)	$B_2(pk, T_1, \text{ch}_1'^*)$ outputs $(\text{rsp}'_1, \text{ch}'_2, \text{rsp}'_2)$
$T := (\text{com}, \text{ch}_1, \text{rsp}_1, \text{ch}_2^*, \text{rsp}_2)$	$T' := (\text{com}', \text{ch}_1'^*, \text{rsp}'_1, \text{ch}'_2, \text{rsp}'_2)$

where T and T' are distributed as the output of the HVZK simulator $\mathcal{S}(pk)$ when ch_2^* and $\text{ch}_1'^*$ are chosen uniformly at random from $\text{ChS}_i(1^\kappa)$.

If the ID scheme has the early abort property, then we additionally require that $\mathcal{V}_{\text{early}}(pk, T_1) = 1$ for simulator A and $\mathcal{V}_{\text{early}}(pk, (\text{com}', \text{ch}_1'^*, \text{rsp}'_1)) = 1$ for simulator B . Note that A_2 is given a ch_2^* and can choose rsp_2 in such a way that T_1 is a prefix for a valid transcript T . Using B_1 and B_2 , we can also produce a valid transcript T' for a given value of $\text{ch}_1'^*$. Together, these properties capture the ability of the attacker to cheat by guessing *either one of* ch_1 or ch_2 correctly. In the MQDSS example, we described the concept of “fixing bad guesses for ch_1 ”, which is captured by the fact that in schemes without early abort, we can use the com output of A_1 as input to B_2 , whereas in schemes with early abort, this might lead to situations where the first three messages $(\text{com}, \text{ch}_1'^*, \text{rsp}'_1)$ of the resulting transcript do not pass $\mathcal{V}_{\text{early}}$.

Generic Attack. The forger is given pk as input, and uses the algorithms (A_1, A_2, B_1, B_2) to create a forgery, as follows. Let m be the message to forge; we assume it is an input to both H_1 and H_2 . Let $\tau^* < r$ be the number of repetitions to guess the first challenge.

1. Using A_1 , compute a triple of the form $(\text{com}, \text{ch}_1^S, \text{rsp}_1)$ for each of the r repetitions (in the case of protocols with early abort, only use A_1 for τ^* repetitions and B_1 for the remaining). Then compute $(\text{ch}_1^{(1)}, \dots, \text{ch}_1^{(r)}) = H_1(\text{com}^{(1)}, \dots, \text{com}^{(r)})$. Repeat this step until $\text{ch}_1^{(i)} = \text{ch}_1^S$ for τ^* repetitions.
2. Fix the value of com for all repetitions so that the ch_1 values do not change. Let R be the set of indices of the τ^* repetitions where $\text{ch}_1^S = \text{ch}_1^{(i)}$. For repetitions $i \notin R$, compute $(\text{rsp}_1^*, \text{ch}_2^S, \text{rsp}_2^*)$ using B_2 and set $\text{rsp}_1^* = \text{rsp}_1$ (from the output of A_1) when $i \in R$. Now compute

$$(\text{ch}_2^{(1)}, \dots, \text{ch}_2^{(r)}) = H_2(\{\text{com}^{(i)}\}, \{\text{ch}_1^{(i)}\}, \{\text{rsp}_1^*(i)\})$$

where $i \in \{1, \dots, r\}$. Repeat until repetitions $i \notin R$ have $\text{ch}_2^{(i)} = \text{ch}_2^S$.

3. For $i \in R$, use A_2 to calculate a valid response rsp_2^* . Output the forgery $(\{\text{com}^{(i)}\}, \{\text{rsp}_1^*(i)\}, \{\text{rsp}_2^*(i)\})$ for $i \in \{1, \dots, r\}$.

We highlight why this attack is only possible for non-interactive proofs. First, in the interactive setting, each try in Step 1 requires interaction with the verifier, which is slow, and may be subject to limits by the verifier. But more importantly, the repeated guesses for ch_2 are not possible while holding the ch_1 values fixed since the verifier will force the prover to restart from the very beginning: all effort to guess the τ^* ch_1 values correctly is lost.

4.1 Cost Analysis

The analysis differs depending on whether the scheme has the early abort property. In both cases, the attack complexity is dependent on the size of the two challenge spaces C_1, C_2 . Let IDS be a 5-pass identification scheme with challenge spaces C_1, C_2 and $|C_1| = q_1, |C_2| = q_2$ and let Dss be the signature scheme derived from r parallel repetitions of IDS by applying a generalized Fiat-Shamir transformation like Construction 1.

Schemes Without Capability for Early Abort. Recall the probability $P_1(\tau, r, q)$ of guessing at least τ of r challenges with a challenge space of size q each correctly, as given per Eq. 1. The expected cost of our attack on Dss is given by

$$\text{Cost}_{\text{non-abort}}(r) = \frac{1}{P_1(\tau^*, r, q_1)} + q_2^{r-\tau^*},$$

where τ^* is the optimal number of repetitions to attack in the first challenge, given by

$$\tau^* = \arg \min_{0 \leq \tau \leq r} \frac{1}{P_1(\tau, r, q_1)} + q_2^{r-\tau},$$

minimizing the overall cost of the attack.

Schemes with Capability for Early Abort. The cost of our attack on Dss is given by

$$\text{Cost}_{\text{abort}}(r) = q_1^{\tau^*} + q_2^{r-\tau^*},$$

where τ^* is the optimal number of repetitions to attack in the first challenge, given by

$$\tau^* = \arg \min_{0 \leq \tau \leq r} q_1^\tau + q_2^{r-\tau},$$

again minimizing the overall cost of the attack.

Following the derivation of the attack costs, the number of parallel repetitions r of the underlying identification scheme IDS needed to achieve a security level of κ bits is given by selecting the minimum value of r such that the corresponding cost function $\text{Cost}(r) \geq 2^\kappa$.

4.2 Discussion

Benefit of Early Abort. We can now quantify the security benefit of protocols with an early abort functionality in some specific examples.² If MQDSS were instead based on a (hypothetical) proof protocol with early abort, the number of parallel repetitions required for 128-bit security would be 153, rather than 184. This is less than half of the increase from 135 (the choice of r given by the ϵ^r -heuristic), motivating the design of a 5-round proof protocol for MQ with early abort; one such protocol is MUDFISH [6], which does result in significantly shorter signatures. Similarly, if the five-round variant of Picnic described in Sect. 5.1 did not have the early abort property, the number of required online phases for 128-bit security is 50 rather than 43, increasing signature size by roughly 1.16x. Thus we find that having the early abort property is a desirable goal for designers of five-round proof protocols, if it does not add additional costs to the protocol itself.

Unbalanced Size of the Challenge Spaces. An interesting observation is the fact that if the two challenge spaces are not of equal size, the attack complexity increases, as an attacker cannot divide the work evenly between the two phases. MQDSS is an example of this, as one challenge space is of size 31 and the second one of size 2, meaning an attacker has to spend more effort guessing the first challenge. However, to get the best attack complexity, the attacker wants to spend an equal amount of work in both phases, meaning attacking fewer rounds in the first challenge than the second one. Again, the five-round variant of Picnic in Sect. 5.1 serves as an example. Both of its challenge spaces are of equal size, and the number of repetitions needs to be doubled to resist the attack, compared to the $\approx 1.4x$ more repetitions needed for MQDSS.

² Because the Cost functions do not have a nice closed form a general comparison appears to be difficult.

Security of $(2n + 1)$ -Round Protocols. We can also ask about similar attacks on proof protocols with more than five rounds. For example, a recently proposed signature scheme that we discuss in Sect. 5.3 has seven rounds, and we can fully generalize the canonical protocol to $2n + 1$ rounds. Selecting the number of parallel repetitions for these protocols is also an interesting question.

However, when considering multiple abort points, analyzing such protocols seems challenging, as there are $n - 1$ places where early aborts are possible, and a specific protocol may have $0 \leq m \leq n - 1$ of $n - 1$ abort points. One could begin by analyzing the worst-case $m = 0$, however, the choice of r would likely be inefficient for protocols with $m > 0$. However, for some protocols, it might be possible to conceptually split them into sub-protocols that have three or five passes and analyze them individually.

5 Application to Other Schemes

In the area of post-quantum signatures, many recent proposals are built using 5-round protocols, made non-interactive using the Fiat-Shamir transformation. We now investigate the applicability of the attack of Sect. 4 on some schemes from the literature.

5.1 Five Round Picnic

Picnic [11] is a second-round candidate in the NIST post-quantum standardization project. It is built using a non-interactive zero-knowledge proof of knowledge, proving knowledge of a secret key of a block cipher. One variant of Picnic is based on the KKW proof system [21].

The KKW proof system is a 5-round interactive protocol based on a multi-party computation protocol with an offline preprocessing phase. In the first round, the prover commits to M executions of the offline phase of the MPC protocol and then gets challenged to open all but one of them. In the third round, the prover then uses the unopened offline phase to execute an online phase for N parties and commits to all of their states and subsequently gets challenged to open all but one of the internal states. Based on the $N - 1$ privacy property of the MPC protocol, the protocol is zero-knowledge and has a soundness error of $\max\{\frac{1}{M}, \frac{1}{N}\}$. However, in Picnic, the scheme is collapsed into a 3-round protocol, as described in Sect. 1.1. In [31], the authors discussed that the 5-round protocol could offer different performance tradeoffs, but also remarked that the soundness calculation changes since “both challenges have to be sufficiently large”. In this section we apply our analysis to choose concrete parameters for the 5-round variant of Picnic, and find that the 3-round variant is indeed preferable.

Cheating Strategy for the Picnic2 Zero-Knowledge Proof. For the attack to work, a cheating signer needs to be able to cheat in either of the two phases of the zero-knowledge proof. In detail, for the KKW proof system, this means either cheating in the pre-preprocessing phase by producing invalid multiplication triples or

cheating in the online phase by sending wrong messages. Both approaches allow the prover to flip the output of arbitrary AND gates in the circuit, if not detected by the verifier. A cheating prover, given a plaintext-ciphertext pair from a target public key, can therefore select a random secret key and start the encryption with the plaintext and change AND gates during the circuit evaluation until the output matches the ciphertext.

Based on the soundness error of the interactive version of 5-round Picnic ($\max\{\frac{1}{M}, \frac{1}{N}\}$, where M is the number of preprocessing phases and N is the number of parties in the online phase) it is optimal to set both of them to be equal. One choice used by [21] is 64, since this fits register widths for modern CPUs, allows for a performant bit-sliced implementation, and provides a good tradeoff between proof size and runtime. To achieve a soundness error of $< 2^{-128}$, one needs $\tau = 22$ parallel repetitions in the interactive version of the protocol. However, applying the straightforward Fiat-Shamir transformation as shown in Construction 1 enables our attack.

Since the protocol in [21] is a commit-and-open style protocol, it has the property of *early abort* (guessing the wrong challenge in the second message cannot be hidden later on). Therefore we need to choose the repetitions to attack in each phase from the start.

The complexity of the attack on 5-round Picnic is $M^{\tau^*} + N^{\tau-\tau^*}$, where τ^* repetitions are attacked in the first challenge. The optimum number of repetitions τ^* which is equal to $\approx \tau/2$, since both challenge spaces are of equal size. For the specific choice of $M = N = 64$, the total number of parallel repetitions required for an attack complexity of greater than 2^{128} random oracle calls is therefore $\tau = 43$.

In contrast to the collapsed 3-round variant, which needs 343 offline phases and the same number of online phases, this 5-round variant needs $43 \cdot 64 = 2752$ offline phases and 43 online phases. We give the performance characteristics of the 3-round (Picnic2-*) and 5-round (Picnic2-5-*) variants in Table 2. Observe that even though the number of online phases that need to be simulated is lower in Picnic2-5, this is only true during signing, as during verification this number is actually higher in the 5-round variant. Furthermore, the number of offline phases and, more importantly, the hashing costs associated with this phase are much higher in the 5-round variant. Even though we did not implement the 5-round variant, we conclude based on this evidence that the 5-round variant has slower signing and verification times. With regards to signature size, the maximum signature size for the 5-round variant is given (in the notation of [21]) by

$$\tau \cdot ((\lceil \log_2(M) \rceil + \lceil \log_2(N) \rceil) \cdot \kappa + 2 \cdot |\mathcal{C}| + 3\kappa + |\text{ch}_1| + |\text{ch}_2|) .$$

In all cases, this leads to larger signature sizes than the three round variants, confirming the choice made in [21] to collapse the protocol to three rounds.

Table 2. Comparison of Picnic2 using the 3- and 5-round variants of the underlying proof system at the three NIST security levels. Picnic2 numbers from [31].

Instance	# offline phases sign (verify)	# online phases sign (verify)	max. signature size [KiB]
Picnic2-L1-FS	343 (316)	343 (27)	13.47
Picnic2-5-L1-FS	2752 (2709)	43 (43)	16.46
Picnic2-L3-FS	570 (531)	570 (39)	29.05
Picnic2-5-L3-FS	4032 (3969)	63 (63)	36.17
Picnic2-L5-FS	803 (753)	803 (50)	53.45
Picnic2-5-L5-FS	5440 (5355)	85 (85)	63.75

5.2 PKP-Based Signature Scheme

In [8], the authors proposed a digital signature scheme based on the Permuted Kernel Problem (PKP) [28]. Since the underlying identification scheme is a 5-pass scheme and the transformation into a signature scheme is using Construction 1 while inheriting all security proofs from the original MQDSS paper [12], it is susceptible to the same attack.

In fact, a pre-print of [8] originally chose the number of parallel repetitions r using the ϵ^r -heuristic, however, it was later revised to account for our attack and use larger parameters. We shortly summarize the parameters of the scheme and show using the formulae in Sect. 4 that the parameters as proposed in the most recent version of [8] are secure.

In PKP-DSS, the size of the first challenge is based on the size of the underlying prime field (excluding 0). Like MQDSS, the second challenge is a binary choice, and the identification scheme does not have the property of early abort. Therefore, we use the same formula as in MQDSS and arrive at 156, 228, and 289 parallel repetitions for their L1, L3, and L5 security levels, respectively. Note that the number of parallel repetitions for the L1 and L3 security levels is lower than the parameters given in [8] (157 and 229, respectively); this might be due to the authors weighting of the cost of the two phases slightly differently.

5.3 LegRoast

LegRoast and PorcRoast [7] are two new proposals for post-quantum secure signature schemes. Our attack is not directly applicable to these schemes, but it is interesting to see why, as they are based on 7-round proof protocols.

The schemes work by proving knowledge of the secret key of evaluations of the Legendre-PRF, in a similar fashion to Picnic, which uses LowMC as a one-way function. The scheme has some more differences to Picnic signatures: since the PRF only outputs a single bit, it needs many different evaluations of the PRF to achieve the needed soundness, however, this would lead to large signatures. Therefore a relaxed notion is used, where the prover proves that B of the total

L evaluations are correct under his secret key. Additionally, instead of using a cut-and-choose construction for the MPC-in-the-head preprocessing step, they use a method based on sacrificing multiplication triples by Baum and Nof [3].

The prover uses a 7-pass identification scheme, where the first challenge selects the subset B of evaluations to prove, the second challenge is for the sacrificing step of the MPC protocol, and the third challenge selects one of N parties to reveal for verification. However, the challenge space of the second challenge is about 128 bits and is therefore much bigger than the third challenge space (which ranges from $N \in \{16, 64, 256\}$). As already shown in [3], this essentially means an adversary gains a negligible advantage when trying to guess the second challenge, and the overall attack complexity is not reduced by attacking this phase. The parameters of LegRoast are thus chosen in a way that rules out attacks that split the work between the first and last phase.

6 Conclusion

In this work, we have shown forgery attacks against a class of signature schemes built from five-pass ID schemes and the Fiat-Shamir transform, highlighting the importance of concrete parameter selection. Our analysis gives designers an accessible way to choose the number of parallel repetitions to meet a given security requirement.

An interesting conclusion for the two schemes we investigated in detail, MQDSS and Picnic, is that initially, the 5-pass variants look more attractive in terms of runtime and signature size, but once accounting for this attack, the 3-pass variant becomes more efficient. In addition to being more well analyzed, this is another reason to prefer 3-round ID schemes.

We did not investigate some of the schemes mentioned in Sect. 1.1, this may be interesting future work. Additionally, with some recent practical 7-round protocols being proposed [3, 7], generalizing our attack beyond five rounds may also be interesting. Finally, our classification of protocols that are vulnerable to this type of attack could be improved, as the properties we used (early abort and piecewise simulatability) are non-standard. Perhaps these properties can be related to existing and more well-studied properties.

Acknowledgments. D. Kales was supported by the European Union’s Horizon 2020 research and innovation programme under grant agreement n°871473 (KRAKEN). D. Kales was additionally supported by iov42 Ltd.

References

1. Aguilar, C., Gaborit, P., Schrek, J.: A new zero-knowledge code based identification scheme with reduced communication. In: 2011 IEEE Information Theory Workshop, pp. 648–652. IEEE (2011)
2. El Yousfi Alaoui, S.M., Dagdelen, Ö., Véron, P., Galindo, D., Cayrel, P.-L.: Extended security arguments for signature schemes. In: Mitrokotsa, A., Vaudenay, S. (eds.) AFRICACRYPT 2012. LNCS, vol. 7374, pp. 19–34. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31410-0_2

3. Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12110, pp. 495–526. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45374-9_17
4. Bellare, M., Impagliazzo, R., Naor, M.: Does parallel repetition lower the error in computationally sound protocols? In: 38th FOCS, pp. 374–383. IEEE Computer Society Press, October 1997. <https://doi.org/10.1109/SFCS.1997.646126>
5. Bettaieb, S., Schrek, J.: Improved lattice-based threshold ring signature scheme. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 34–51. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_3
6. Beullens, W.: Sigma protocols for MQ, PKP and SIS, and fishy signature schemes. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 183–211. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_7
7. 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
8. Beullens, W., Faugère, J.-C., Koussa, E., Macario-Rat, G., Patarin, J., Perret, L.: PKP-based signature scheme. In: Hao, F., Ruj, S., Sen Gupta, S. (eds.) INDOCRYPT 2019. LNCS, vol. 11898, pp. 3–22. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35423-7_1
9. Cayrel, P.-L., Lindner, R., Rückert, M., Silva, R.: A lattice-based threshold ring signature scheme. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 255–272. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14712-8_16
10. Cayrel, P.-L., Véron, P., El Yousfi Alaoui, S.M.: A zero-knowledge identification scheme based on the q -ary syndrome decoding problem. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 171–186. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19574-7_12
11. Chase, M., et al.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 1825–1842. ACM Press, October/November 2017. <https://doi.org/10.1145/3133956.3133997>
12. Chen, M.-S., Hülsing, A., Rijneveld, J., Samardjiska, S., Schwabe, P.: From 5-pass MQ-based identification to MQ-based signatures. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 135–165. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_5
13. Chen, M.S., Hülsing, A., Rijneveld, J., Samardjiska, S., Schwabe, P.: MQDSS specifications (March 2019), version 2.0. mqdss.org/files/MQDSSVer2.pdf
14. Chen, S., Zeng, P., Choo, K.K.R., Dong, X.: Efficient ring signature and group signature schemes based on q -ary identification protocols. *Comput. J.* **61**(4), 545–560 (2018)
15. Dambra, A., Gaborit, P., Roussellet, M., Schrek, J., Tafforeau, N.: Improved secure implementation of code-based signature schemes on embedded devices. *Cryptology ePrint Archive*, Report 2014/163 (2014). <http://eprint.iacr.org/2014/163>
16. El Yousfi Alaoui, S.M., Cayrel, P.-L., El Bansarkhani, R., Hoffmann, G.: Code-based identification and signature schemes in software. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L. (eds.) CD-ARES 2013. LNCS, vol. 8128, pp. 122–136. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40588-4_9

17. 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
18. Håstad, J., Pass, R., Wikström, D., Pietrzak, K.: An efficient parallel repetition theorem. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 1–18. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11799-2_1
19. 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, June 2007. <https://doi.org/10.1145/1250790.1250794>
20. Kales, D., Zaverucha, G.: An attack on some signature schemes constructed from five-pass identification schemes. IACR Cryptol. ePrint Arch., p. 837 (2020). <https://eprint.iacr.org/2020/837>. (Full version of this paper)
21. 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, October 2018. <https://doi.org/10.1145/3243734.3243805>
22. Kiltz, E., Loss, J., Pan, J.: Tightly-secure signatures from five-move identification protocols. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10626, pp. 68–94. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70700-6_3
23. Kobitz, N., Menezes, A.: Critical perspectives on provable security: fifteen years of “another look” papers. IACR Cryptol. ePrint Arch. **2019**, 1336 (2019)
24. National Institute for Standards and Technology: Post-quantum cryptography: Call for proposals (2016). <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf>
25. Pietrzak, K., Wikström, D.: Parallel repetition of computationally sound protocols revisited. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 86–102. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_5
26. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_33
27. Sakumoto, K., Shirai, T., Hiwatari, H.: Public-key identification schemes based on multivariate quadratic polynomials. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 706–723. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_40
28. Shamir, A.: An efficient identification scheme based on permuted kernels (extended abstract). In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 606–609. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_54
29. Stern, J.: A new identification scheme based on syndrome decoding. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 13–21. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_2
30. Stern, J.: A new paradigm for public key identification. IEEE Trans. Inf. Theory **42**(6), 1757–1768 (1996)
31. The Picnic Design Team: The Picnic signature scheme design document, March 2019, version 2.0. <https://microsoft.github.io/Picnic/>



Energy Analysis of Lightweight AEAD Circuits

Andrea Caforio, Fatih Balli, and Subhadeep Banik^(✉)

LASEC, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
{andrea.caforio,fatih.balli,subhadeep.banik}@epfl.ch

Abstract. The selection criteria for NIST’s Lightweight Crypto Standardization (LWC) have been slowly shifting towards the lightweight efficiency of designs, given that a large number of candidates already establish their security claims on conservative, well-studied paradigms. The research community has accumulated a decent level of experience on authenticated encryption primitives, thanks mostly to the recently completed CAESAR competition, with the advent of the NIST LWC, the de facto focus is now on evaluating efficiency of the designs with respect to hardware metrics like area, throughput, power and energy.

In this paper, we focus on a less investigated metric under the umbrella term lightweight, i.e. energy consumption. Quantitatively speaking, energy is the sum total electrical work done by a voltage source and thus is a critical metric of lightweight efficiency. Among the thirty-two second round candidates, we give a detailed evaluation of the ten that only make use of a lightweight or semi-lightweight block cipher at their core. We use this pool of candidates to investigate a list of generic implementation choices that have considerable effect on both the size and the energy consumption of modes of operation circuit, which function as an authenticated encryption primitive.

In the second part of the paper, we shift our focus to threshold implementations that offer protection against first order power analysis attacks. There has been no study focusing on energy efficiency of such protected implementations and as such the optimizations involved in such circuits are not well established. We explore the simplest possible protected circuit: the one in which only the state path of the underlying block cipher is shared, and we explore how design choices like number of shares, implementation of the masked s-box and the circuit structure of the AEAD scheme affect the energy consumption.

Keywords: Energy · Power · Lightweight cryptography · AEAD · Block ciphers · Unrolling · Hardware · Logic synthesis

1 Introduction

Applications running on resource-constrained devices generally require a decent level of protection regarding their communication layer, even though the allocated budget for security tends to be sparse. It presents itself in the form of constraints over few metrics, such as circuit size, energy consumption, or latency;

and prioritization among them depends on the particular application and the device in question. Sensor networks, medical implants, smart cards, and Internet-of-Things are a selection of applications where either one or few of these metrics play a key role.

These constraints spurred multiple lines of research in the crypto community, one mainly focusing on realizing the standardized symmetric primitives in a more lightweight manner. For instance, reducing the circuit-size of AES has been extensively studied [6, 21, 23]. On a separate line of research, bootstrapping new primitives from scratch is taken as an alternative and is possibly more fruitful approach to obtain symmetric primitives with better lightweight characteristics. This justifies why the literature has seen a large number of new block ciphers such as PRESENT [13], SKINNY [11], and GIFT [10], to name only a few. There are even some attempts to discover new techniques to improve *lightweightness* of these new block ciphers [3, 21]. As block ciphers alone are not ready-to-use primitives but rather need to be wrapped in a mode of operation, a group of candidates in NIST LWC utilize these lightweight block ciphers to attain an authenticated encryption (AE) primitive, i.e. the ten candidates on which this paper focuses [1].

Banik et al. [5] finally presented a model that captures the energy consumption of a block cipher in terms of r , where r denotes the number of unrolling in an implementation. For many ciphers, including AES, their model verifiably predicts that the energy-optimal choice is $r = 1$, where for some lighter block ciphers, such as PRESENT, the optimal point shifts to $r = 2$. However, as stated before, block ciphers usually are not ready-to-use primitives and must be wrapped within a mode of operation. Therefore, the effects of additional circuitry to energy consumption remains unanswered.

1.1 Contributions and Organization

1. We explore the effects of clock-gating, r -round unrolling, fully-unrolling and inverse-gating techniques to deduce the architectural design choices that lead to the most energy efficient implementations. We look at each candidate individually and identify optimal circuit configurations that would reduce the energy consumption of AEAD circuit. The large number of implementations helps us make broader observations regarding energy efficiency in AEAD modes instantiated with lightweight block ciphers.
2. In parallel to the first effort, we provide a fair evaluation of the aforementioned candidates from NIST LWC. The data we obtain shows how each candidate fares, when implemented with the similar approach.
3. In partially unrolled circuits, we demonstrate that the optimal choice of r boils down to two factors; the complexity of the core cipher and the complexity of the surrounding mode of operation circuitry. Whereas the optimal choice for block ciphers is typically $r \in \{1, 2\}$, for full AE circuits we experimentally show that this becomes $r \in \{2, 3\}$.
4. In the last part of the paper we move to threshold implementations that provide security against power analysis attacks. Although there have been

many papers that optimize the circuit area of these circuits [11, 25], there have not been many papers that look at the energy consumption of these circuits as an optimizable metric. We look at both 3-share and 4-share threshold circuits, and look at factors like number of shares, *decomposability* of s-boxes that affect the energy consumption of such circuits.

The paper unfolds as follows. Section 2 reiterates known energy-reduction techniques and lays out a common interface and test bench for all implementations. In Sect. 3, we briefly introduce the chosen schemes alongside their internal block ciphers and detail their implementations. Section 4 evaluates the effects of the individual design choices on the schemes and extends Banik et al.’s energy model of block ciphers to modes of operations in the chosen authenticated encryption algorithms. We also elaborate on the obtained energy measurements and chart the results. In Sect. 5, we turn our attention to first order threshold implementations of the AEAD schemes. We conclude our paper with the takeaway claims for designers and implementors in Sect. 6.

2 Preliminaries

To guarantee fair conditions in our evaluation we unified our implementations under a common interface. Our hardware API is designed to be simple, as it assumes that the associated data and message bits are properly padded so that they only consists of multiple blocks. This padding must be done according to the individual specification of the AE scheme, before the AE operation is initiated in the circuit. Then our AEAD implementations can be used in all possible configurations (e.g. partial blocks, no authenticated data or no message blocks) and comply with the exact specification.

Our reasoning for favoring this simpler API (with external-padding) is that it ensures that no significant energy is consumed to handle the API itself. For instance, the CAESAR HW API [18] requires padding to be done by the circuit, which brings a large array of multiplexers and amplifies the energy consumption for each loaded associated data and message block. However, depending on the application, this padding cost can be avoided, e.g. handling padding on a microprocessor that makes the call can be less costly, or the application might not even need padding, if the transmitted data always respects the block sizes. Nonetheless, a preprocessor circuit could be placed before our AE schemes to ensure CAESAR HW API compatibility. The input and output ports of our hardware API are defined in the following way:

- `input_wire` CLK, RST: System clock and active-low reset signal. We distinguish two different clock rates; 10 MHz for the partially-unrolled versions and 5 MHz for the fully-unrolled implementations¹.

¹ The inverse-gating technique uses only the first phase of the clock cycle to compute the full block cipher call, therefore the clock period is doubled to ensure all glitches are stabilized during this clock phase.

- `input_vector` KEY, NONCE: Key and nonce vectors. These signals are stable once the circuit is reset and are kept active during the entire computation.
- `input_vector` DATA: Single data vector from which both associated data and regular plaintext blocks are loaded into the circuit. This choice saves an additional large multiplexer, since all the schemes process associated data and plaintext blocks separately and not in parallel.
- `input_wire` EAD, EPT: Single bit signals that indicate whether there are no associated data blocks (EAD) or no plaintext blocks (EPT). Both signals are supplied with the reset pulse and remain stable throughout the computation.
- `input_wire` LBLK, LPRT: Single bit signals that indicate whether the currently processed block is the last associated data block or the last plaintext block (LBLK), and also whether it is partially filled (LPRT). Both signals are supplied alongside each data block and remain stable during its computation.
- `output_wire` BRDY, ARDY: Single bit output indicators whether the circuit has finished processing a data block and a new one can be supplied on the following rising clock edge (BRDY) or the entire AEAD computation has been completed (ARDY).
- `output_wire` CRDY, TRDY: Single bit output indicators whether the CT and TAG ports will have meaningful ciphertext and tag values starting from the following rising clock edge.
- `output_vector` CT, TAG: Separate ciphertext and tag vectors. This again saves an additional multiplexer in schemes where the ciphertext and tag are not ready at the same time, or they appear at different wires.

2.1 Test Bench and Synthesis Options

One of the criteria in the NIST lightweight competition lies in the optimization of the proposed schemes when they are fed with messages as short as eight blocks. Hence, our test bench focuses on supplying the circuit with various input lengths where a single AE call contains at most one associated data block (where we consider each block as 128-bit), along with a random number of message blocks (not more than eight blocks to make it short). The corner cases are also captured by generating inputs with either empty authenticated data or empty message, as well as incomplete last blocks. Each round-based AEAD implementation is run with the same test vector, where the length ratio between authenticated data blocks and message blocks is roughly one to eight. This is summarized in Table 1.

Another point to consider is that the power dissipation and energy consumption of an ASIC circuit is highly sensitive to the actual silicon technology it is implemented, as well as generic optimization techniques available to the development kit. The RTL synthesizer (in our case Synopsys Design Vision v2019.03) can also bring a significant change in the results depending on the compilation flags. In order to isolate variations in energy consumption caused by these, besides using the same technology (TSMC 90 nm), we maintain compilation options consistent when comparing candidates. Although these variations

are not likely to change the final ordering of candidates, it makes it difficult to reproduce results if subtle details are not reported.

In order to keep things simple, we used them in the following combinations: The `compile_ultra` option instructs the compiler to perform an all-in-one, computationally intensive optimization, during which boundaries between components are removed and the whole design is considered as one large circuit. This provides better results for r -round unrolled implementations, but becomes time consuming and works poorly as r grows larger. Therefore we do not use this option with fully-unrolled implementations. The command `compile -exact_map -area_effort high` command essentially ensures that sequential elements are not touched, and that Synopsys favors area as a metric to improve (a common flag used by designer, but not of vital importance in our case). This combination is ideal for unrolled circuits, as the area by default is already quite large and there are possibly many optimizations to perform. A third combination `compile -no_autoungroup` is used only to obtain results in Sect. 4.2. This flag instructs Synopsys not to remove the boundaries between components at lower level, so that we can obtain power consumption of each individual element, and compute necessary parameters in our model. For clock-gating implementations, we first compiled clock-gating circuitry and then used `set_dont_touch` option to ensure that Synopsys does not try to optimize it later, as it may lead to timing violations.

Table 1. Synthesis options, and the size of test vectors

Implementation	Synopsys Compilation Flags	# AD blocks	# Msg blocks
r -Round AE	<code>compile_ultra</code>	535	4278
Fully-unrolled	<code>compile -exact_map -area_effort high</code>	28	207
Section 4.2	<code>compile_ultra -no_autoungroup</code>	535	4278
Threshold	<code>compile_ultra</code>	535	4278

For all results reported in the paper, we maintained the following design flow. The design was implemented at RTL level. A functional verification of the VHDL code was then done using *Mentor Graphics ModelSim*. Thereafter, *Synopsys Design Compiler* was used to synthesize the RTL design using the compile options in Table 1, and post-synthesis correctness is verified with *Synopsys VCS MX Compiled Simulator*. The switching activity of each gate of the circuit was collected by running post-synthesis simulation. The average power was obtained using *Synopsys Power Compiler*, using the back annotated switching activity. The “energy per processed block” metric was then computed as the product of the average power and the total time taken to process x number of blocks, divided by x itself.

Clock-Gating: Clock-gating describes a general power-reduction technique that aims to limit the switching activity of register banks. A classic, non-gated flip-flop is continuously charged and discharged by the system clock which results in

wasted activity during periods when the flip-flop needs to preserve its content for multiple cycles. The clock signal in a clock-gated register is artificially held constant through additional logic during these constant phases.

Inverse-Gating: In a sequential arrangement of round function circuits, the glitches generated in the first computation, until a stable value is reached, are amplified in the subsequent round function calls, which is responsible for most of the dynamic power consumption of the entire implementation. Banik et al. suggested round-gating as an effective countermeasure against the propagation of glitches between the round functions [7]. The technique saw a revision in 2018, coined *inverse-gating*, through an addendum by the same authors [8]. In broad terms, inverse-gating suppresses the glitches between round functions by inserting AND gates on the critical path which are then activated by a delayed clock signal. Preferably, the delay time should be at least as big as the signal latency at each round function output. Figure 1 depicts an inverse-gated unrolled arrangement.

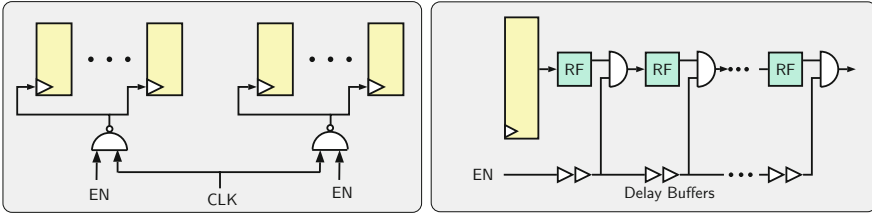


Fig. 1. Partitioned clock-gated register (left), fully-unrolled inverse-gated round function (right).

3 Implementations

Out of the 32 remaining candidates in the second round of the NIST lightweight competition we singled out ten schemes that are bootstrapped either directly via lightweight block ciphers or variants of them. Five out of the ten schemes are directly instantiated with the GIFT block cipher [10] or through a slightly adapted tweakable alteration. Three other schemes are based on the SKINNY block cipher [11] or a forked version of it. Finally, the Pyjamask and SATURNIN AEAD schemes deploy their own dedicated substitution-permutation networks of the same names. Table 2 lists all investigated schemes alongside their internal block cipher.

3.1 r-Round Unrolled

The sequential placement of multiple round function circuits allows the computation of several rounds during a single clock cycle. This results in fewer

Table 2. AEAD Schemes Based on Lightweight Block Ciphers. CG denotes clock-gated and IG inverse-gated implementations.

Scheme	Block Cipher	Reference	Best Implementation
GIFT-COFB	GIFT-128	[9]	2-Round-CG
SUNDAE-GIFT	GIFT-128	[4]	3-Round
HYENA	GIFT-128	[16]	2-Round-CG
LOTUS-AEAD	TWE-GIFT-64	[15]	3-Round-CG
LOCUS-AEAD	TWE-GIFT-64	[15]	3-Round-CG
SKINNY-AEAD	SKINNY-128-384	[12]	Unrolled-IG
Romulus	SKINNY-128-384	[19]	2-Round
ForkAE	ForkSkinny	[2]	2-Round-CG
Pyjamask	Pyjamask-128	[17]	Unrolled-IG
SATURNIN	SATURNIN	[14]	Unrolled-IG

required cycles to complete one encryption, i.e. in an r -round partial unrolling setting a block cipher composed of R rounds can be computed in $\lceil \frac{R}{r} \rceil$ cycles. The adverse effects of unrolling include a larger overall circuit area and an increased signal delay across the circuit. Nevertheless, as shown by Banik et al. [5], partial unrolling can reduce the energy consumption of certain (especially lightweight) block ciphers noticeably. In broad terms, it is possible to quantify the total amount of consumed energy E as a quadratic polynomial function of the unrolling factor r such that $E = (Ar^2 + Br + C) \left(\lceil 1 + \frac{R}{r} \rceil \right)$, where A, B and C represent energy values depending on the internal switching activity of the block cipher such as registers, multiplexers and arithmetic logic. Hence, if the block cipher is on the lighter side, E can be minimized for $r \geq 2$, on the other hand complex and heavy circuits such as AES incur large constants A, B and C where E is only minimized for $r = 1$. Using partial r -round unrolling the round function and key expansion circuits can be replicated r times and connected through data paths where the output of the last replicated circuit is stored in the state and key registers. Special care has to be taken when $r \nmid R$, here the ciphertext will not be produced by the last replicated instance but it must come from an intermediate computation as can be seen in Fig. 2.

3.2 Fully-Unrolled

In a fully-unrolled setting we have $r = R$, i.e. an entire encryption is performed in a single clock cycle. Such a configuration results in large combinatorial and latency-heavy circuit. However, state registers that store intermediate results, as previously seen for partially-unrolled block ciphers, are not needed anymore. The propagation and the subsequent amplification of glitches between the round function circuits cause a large spike in terms of energy consumption for which inverse-gating is an effective remedy. In particular, the overall reduction in energy can be as large as 90% for certain schemes as demonstrated in [8].

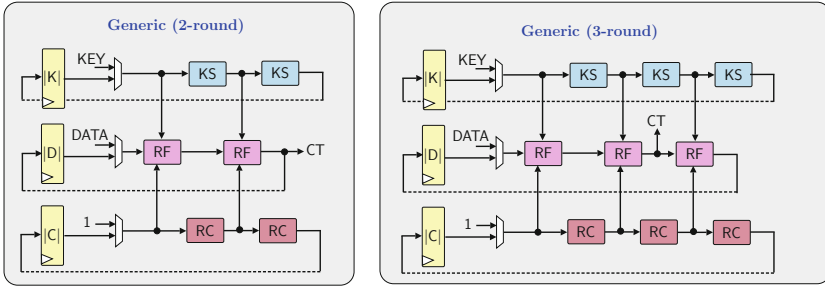


Fig. 2. r -round partial unrolling of a generic block cipher consisting of an internal state, round keys and round constants for $r = 2$ (left) and $r = 3$ (right).

4 Effects of Design Choices

4.1 Clock Frequency

Note that it has already been shown in numerous papers [5,7,22] that in low leakage environments, at high enough frequencies, the total energy consumption of a circuit is independent of clock frequency since it is the measure of total circuit glitch. To provide more evidence for this we constructed a typical circuit for a round based implementation of AES-128, in the TSMC 90 nm library and measured the energy per encryption value at 4 different frequencies. The results are summarized in Fig. 3. The *Energy vs Frequency* plot on the left clearly suggests that for frequencies larger than 10 MHz, the energy consumption is more or less constant.

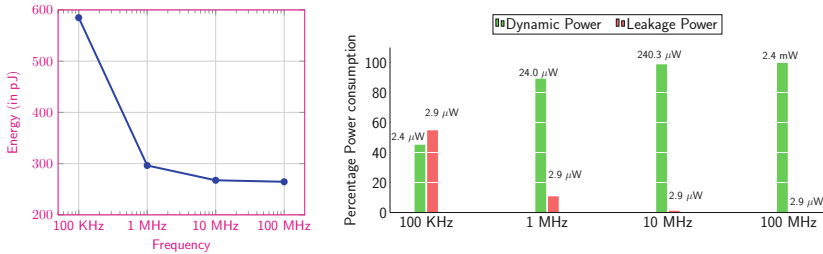


Fig. 3. Variation of energy with frequency (left), percentage contribution of the dynamic, leakage component of the power at different frequencies (right).

Why does this happen? The total power consumption in a CMOS circuit comes from two components **a)** dynamic and **b)** leakage. Dynamic power is consumed due to the charging and discharging of the capacitive nodes of the transistors of the circuit. Every $0 \rightarrow 1/1 \rightarrow 0$ transition, as well as every transient glitch contributes to this type of power consumption. On the other hand, leakage

power is mainly due to the sub-threshold leakage current, which is the drain-source current in a CMOS gate when the transistor is off, and other continuous currents drawn by the power source. It is generally well known that, the leakage component of the power drawn by a circuit is generally independent of the frequency of operation of the circuit and only varies as its total silicon area. Also the dynamic component of the power consumption varies directly as the clock frequency of the circuit and hence inversely as the clock period. Since the total physical time to complete any operation, all other things being the same, varies directly as the clock period, the dynamic component of the energy consumption (product of dynamic power and total time) is generally constant with respect to change in clock period or frequency. Therefore, at higher frequencies the dynamic energy of the same circuit remains a constant as the contribution of the leakage energy (product of the frequency independent leakage power and the physical time taken) becomes lesser and lesser. This was what led [5, 22] to conclude that at high frequencies the total energy (sum of dynamic and leakage energies) consumption of block ciphers is more or less a constant and frequency-independent. All the above facts are borne out by right hand plot in Fig. 3, which breaks down the percentage contributions of the dynamic/leakage power at different frequencies of the same AES circuit. The dynamic component indeed scales as the frequency and the leakage component remains constant at $2.9 \mu\text{W}$. As a result at higher frequencies, the contribution of the leakage part becomes more and more insignificant. In fact at 10 MHz it is less than 1.2%.

Note that for libraries with standard cells composed of transistors of lower feature size, the leakage power is significant even at 10 MHz. Typically, a 15 nm library will have leakage power many orders more than a 90 nm library. For such libraries, a similar exercise of comparing dynamic energy must be done at frequencies much higher than 10 MHz, (for the Nangate 15 nm library for example a clock frequency of around 5–10 GHz may be required). Once this is done, the results reported in 90 nm libraries, can be seamlessly reproduced in 15 nm or lower feature size libraries.

4.2 Optimal Unrolling

In Sect. 3.1, we had briefly mentioned that the energy consumed by an r -round unrolled block cipher as described in [5] is given as $E = (Ar^2 + Br + C) \left(\lceil 1 + \frac{R}{r} \rceil\right)$. The $(Ar^2 + Br + C)$ term is actually the average power consumed by the circuit and is typically output by any standard power compiler engine after inspecting either the switching statistic of every node or the value change dump file that records all the signal transitions in the circuit in a given time period. The term is then multiplied with $\left(\lceil 1 + \frac{R}{r} \rceil\right)$ to produce the energy consumed. Consider an example from [5]. The authors had estimated that in the STM 90 nm process, the energy consumption of an unrolled implementation of PRESENT followed the expression $(3.15 + 1.40r + 0.795r^2) \cdot \left(1 + \lceil \frac{32}{r} \rceil\right)$ pJ. It is elementary to see that $r = 2$ is the minimum of this expression, as for $r = 1, 2, 3$ the expression evaluates to 176.85, 155.21, 174.06 pJ respectively. Now consider PRESENT used in a mode of operation that employs, some other operations like doubling over a

finite field, writing on a register, XORing values etc., i.e. operations that increase the constant term in the quadratic expression.

Suppose that to encrypt 8 blocks of plaintext using the mode requires 10 calls to the block cipher and the extra energy per cycle consumed in the *unrolling-independent* operations is α pJ per cycle. Let's say the mode requires $(1 + \lceil \frac{32}{r} \rceil)$ cycles for the computation (10 block cipher calls). This makes the energy expression for the mode $E(r) = [\alpha + (3.15 + 1.40r + 0.795r^2)] \cdot (1 + \lceil \frac{32}{r} \rceil) \cdot 10$ pJ. If $\alpha \approx 4$ or more, it is now clearly visible that the minima of this expression is $r = 3$, since it evaluates to 3.084, 2.232, 2.220, 2.292 nJ, for $r = 1, 2, 3, 4$. Thus although, the block cipher itself may be energy-optimal at a particular degree of unrolling, it does not necessarily imply that the mode will also be energy-optimal at the same degree. In fact, this is a phenomenon we have observed for 3 lightweight modes of operation SUNDIAE-GIFT, LOTUS-AEAD, LOCUS-AEAD. The modes of operation are all based on the GIFT block cipher. Although the block cipher itself is energy-optimal at $r = 2$, the modes are optimal at $r = 3$. Note that this optimum is subject to other operating parameters like choice of library, or the level of compile time optimization of the circuit etc., but all other things remaining same, this observation stands.

To illustrate the point further, we experimented with 3 non-clock-gated lightweight modes of operation: GIFT-COFB, SUNDIAE-GIFT, and LOTUS-AEAD, all of which are instantiated with some version of the GIFT cipher. Table 3 illustrates the power consumption breakdown of individual components of the 1, 2 and 3-round unrolled implementations of the modes². Note that the 3-round unrolled implementation uses an additional multiplexer to filter signals. Since the total number of rounds in both GIFT-64/128 are not multiples of 3, the signals used to update the state after the execution of 3 rounds in each clock cycle, and the final output of the block cipher are to be tapped from different circuit nodes and hence the need for an extra mux. Note that for this particular implementation, GIFT-COFB and SUNDIAE-GIFT attain optimal energy configuration at $r = 2$, whereas LOTUS-AEAD optimizes at $r = 3$. Take the case of LOTUS-AEAD, in which as the degree of unrolling r increases, the power consumption contribution of the terms depending on r , which are the individual round functions and the incremental components of the state/key registers, increase moderately. This is in contrast to the constant power consumption sources like control system, writing values to various registers and consumption of other gates, all of which increase the constant term in the power consumption. Hence the energy consumed to process eight blocks of plaintext and one block of associated data is around 10.88, 7.20, 6.15 nJ for $r = 1, 2, 3$ respectively. This is not the case for both of these particular implementations of GIFT-COFB or SUNDIAE-GIFT, and hence the optimum point remains at $r = 2$.

² To obtain these figures which illustrate the power consumption of individual circuit elements, we used a different compile directive to the circuit compiler, hence the figures are slightly different from the optimal energy figures tabulated in Table 4.

Table 3. Breakdown of power consumptions of three lightweight modes.

Candidate	Impl.	μW									
		Key Reg.	State Reg.	Other Regs.	Multiplier	Control	RF1	RF2	RF3	Extra Mux.	Total
GIFT-COFB	1-Round	19.2	22.5	7.3	–	10.2	12.5	–	–	–	71.1
	2-Round	20.1	36.4	3.1	–	14.5	13.8	31.1	–	–	119
	3-Round	20.8	34.3	8.1	–	12.4	17.8	33.9	48.7	13	189
SUNDAE-GIFT	1-Round	19.3	28.7	–	3.0	10.8	13	–	–	–	74.8
	2-Round	20.1	36.4	–	3.1	14.5	13.8	31.1	–	–	119
	3-Round	20.8	45.2	–	3.1	12.1	16.4	31.3	46.7	13.4	189
LOTUS-AEAD	1-Round	22	12.3	44.4	–	17.3	9	–	–	–	105
	2-Round	22.1	13.9	52.4	–	23.9	8.8	17.9	–	–	139
	3-Round	22.2	16.7	45.7	–	20.9	11.5	16.4	25.4	6.2	165

4.3 Clock-Gating

We have applied clock-gating technique only for those implementations which contain idle registers, i.e. round-based implementations. These are GIFT-COFB, HYENA, LOTUS-AEAD, LOCUS-AEAD, SKINNY-AEAD, ForkAE, Pyjamask, Romulus and SATURNIN.

In Table 4, we report the number of clock cycles it takes to process the baseline AEAD input, which consists of one authenticated data and eight message blocks, where each block contains 128 bits. As already explained, clock-gating saves energy by preventing unnecessary reloading of registers with the same value, therefore the total energy saving grows proportionally with the total number of clock cycles of an AEAD operation. In other words, the effects of this technique becomes obvious for (1) candidates with more AEAD registers (2) r -round unrolled implementations with small $r \in \{1, 2\}$ as they require more clock cycles. For instance, because 1-round unrolled LOTUS-AEAD implementation lasts 1036 clock cycles, and the design contains a couple of 64-bit registers, this technique saves more than one third in energy. This gap between the implementations are presented in Fig. 4 for LOTUS-AEAD, ForkAE, SKINNY-AEAD.

Therefore, as a rule of thumb, clock-gating is a worthwhile effort if the particular design in question contains large number of flip flops, e.g. registers, that stays frozen for hundreds of cycles.

4.4 Inverse-Gating

The general effect of an inverse-gated fully-unrolled block cipher is a drastic reduction in terms of energy as already demonstrated in [7, 8]. Most of the ten selected AEAD in this paper are thin wrappers around a core block cipher, i.e. additional storage elements but no large combinatorial circuits on the critical path. It thus not surprising that those results extrapolate to the full AEAD construct, however with different magnitudes.

The largest reduction can be observed for SKINNY-AEAD due to its isolated block cipher instance that is directly fed from either the data input or the state registers and whose output glitches are not amplified in a subsequent combinatorial function. Such an arrangement is thus even more energy-efficient than the

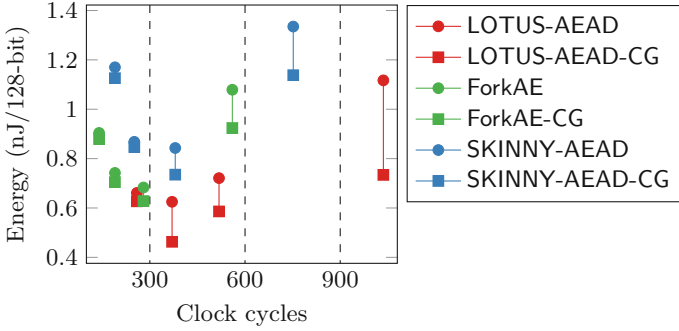


Fig. 4. The energy consumption (per 128-bit block) of round based implementations of LOTUS-AEAD, ForkAE, SKINNY-AEAD with/without clock-gating, in comparison to the number of clock cycles.

partially-unrolled implementations. Similar effects can be noted for the schemes that are based on GIFT or TWE-GIFT whose structures only place relatively lightweight combinatorial functions in front or after the core block cipher. However, the reduced energy does not fully undercut the partially-unrolled implementations. An energy consumption chart of all the fully-unrolled implementations with inverse-gating or without can be seen in Fig. 5.

4.5 Results

Figure 5 (left) charts the optimal energy per 128-bit block value for each r and candidate. The category is dominated by GIFT-COFB and HYENA which both are lightweight in terms of gate count but respond equally well to partial unrolling. The situation is different for the fully-unrolled implementation where inverse-gating equalizes most of the measured values. Figure 5 (right) charts the energy per 128-bit block results for the fully-unrolled variants. A detailed tabulation of all the measurements including gate count, latency and throughput can be found in Tables 4 and 5.

The measurements are reached as follows:

- The latency reports the total number of clock cycles it takes for an AEAD circuit to process 128 bits of authenticated data followed by $8 \times 128 = 1024$ bits of message.
- Throughput of the circuit is calculated by $TP = \frac{9 \times 128}{\text{latency} \times \tau}$ where τ denotes the critical path delay. This is the maximum achievable on this circuit. Further throughput optimizations are possible by instructing Synopsys Design Compiler to recompile the design with additional time constraints, but this falls outside the scope of the paper.
- Average power P_{avg} is directly obtained by the Synopsys Power Compiler. The total energy is computed by $E_{\text{total}} = P_{\text{avg}} \times t$ where t is the time it takes to process the full test vector (see Table 1). Then we divide E_{total} by the number

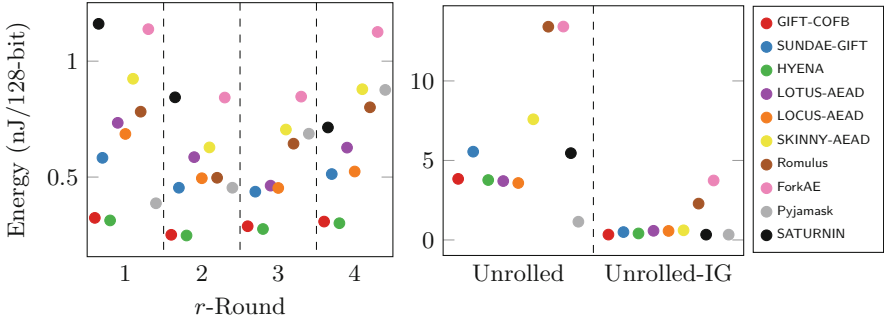


Fig. 5. (left) Energy consumption (nJ/128-bit) comparison chart for the r -round partially-unrolled implementations with $r \in \{1, 2, 3, 4\}$. For each candidate the best obtained energy value obtained through techniques from Sect. 2 is used. (right) Energy consumption (nJ/128-bit) comparison chart for the fully-unrolled implementations with and without inverse-gating.

of processed data blocks (authenticated data and message combined). In the Tables 4–5 below, the flag CG represents circuits designed with the clock-gating technique.

5 Threshold Implementations

The idea of threshold implementations (TI) are based on the concept of multi-party computation and secret sharing, and aims to implement non-linear functions in order to use it as a d -th order DPA countermeasure on a device that leaks information through side channels like power consumption.

It is well known that the minimum number of input shares required to implement the first order TI of a function of algebraic degree w is $w + 1$ [24]. This means that quadratic s-boxes need at least 3 shares and cubic s-boxes need at least 4 shares even for first order TI. However the more the number of shares, we proportionally need to scale up the number of registers and other constituent logic gates in the circuit. Needless to say this comes with proportional scaling up of not only the circuit area but also power and energy consumption of the circuit. Thus at first glance it might appear that, from an energy efficiency point of view, one should rather aim to minimize the number of shares in the circuit.

Most lightweight cryptographic s-boxes are of algebraic degree 3 (e.g. those of PRESENT, GIFT, MIDORI) and hence for a while it was inconceivable to construct a TI of less than 4 shares. However, in [25], the authors showed how to construct 3-share TI of a block cipher cubic s-box. In the paper, the authors presented a 2300 GE 3-share TI of the PRESENT block cipher. The idea is as follows: although the s-box S of PRESENT is cubic, it can be written as $S = F \circ G$, where F and G are quadratic s-boxes. So a 3 shared implementation of the PRESENT s-box can be done by implementing the TI of G and F separated

Table 4. Measurements for the GIFT-COFB, SUNDIAE-GIFT, HYENA, LOTUS-AEAD, LOCUS-AEAD, SKINNY-AEAD, Romulus and ForkAE implementations.

(a) GIFT-COFB						(b) SUNDIAE-GIFT					
Implementation	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (μ W)	Energy (nJ/128-bit)	Implementation	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (μ W)	Energy (nJ/128-bit)
1-Round	400	4710	615.38	69.3	0.363	1-Round	720	3548	430.11	69.4	0.583
1-Round-CG	400	4700	569.17	61.9	0.324	2-Round	360	4313	642.57	107.8	0.454
2-Round	200	5548	1192.55	106.8	0.280	3-Round	252	5136	769.42	147.7	0.437
2-Round-CG	200	5510	952.06	95.5	0.251	4-Round	180	5858	863.70	242.5	0.513
3-Round	140	6372	1211.87	159.0	0.293	Unrolled	18	34571	1145.93	12045.5	5.551
3-Round-CG	140	6311	1172.16	156.2	0.288	Unrolled-IG	18	42419	395.01	1076.7	0.496
4-Round	100	7144	1304.64	237.0	0.314						
4-Round-CG	100	7036	1140.59	232.4	0.308						
Unrolled	10	35735	2015.75	12628.4	3.841						
Unrolled-IG	10	43584	711.15	1107.0	0.337						

(c) HYENA						(d) LOTUS-AEAD					
Implementation	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (μ W)	Energy (nJ/128-bit)	Implementation	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (μ W)	Energy (nJ/128-bit)
1-Round	400	3941	744.19	68.3	0.358	1-Round	1036	6462	223.29	88.08	1.117
1-Round-CG	400	3850	662.07	59.8	0.313	1-Round-CG	1036	6150	223.29	57.9	0.734
2-Round	200	4746	1062.73	97.5	0.256	2-Round	518	6938	358.70	108.3	0.721
2-Round-CG	200	4787	1066.67	94.4	0.248	2-Round-CG	518	6710	376.94	88.10	0.586
3-Round	140	5629	1380.63	151.7	0.280	3-Round	370	7404	431.23	138.3	0.625
3-Round-CG	140	5542	1413.84	149.2	0.276	3-Round-CG	370	7154	441.63	102.2	0.463
4-Round	100	6327	1425.74	227.2	0.301	4-Round	259	7843	481.37	181.5	0.661
4-Round-CG	100	6238	1500.00	232.4	0.307	4-Round-CG	259	6238	539.14	171.9	0.627
Unrolled	10	34988	2045.45	12389.3	3.768	Unrolled	37	19867	819.56	4013.4	3.704
Unrolled-IG	10	49661	711.02	134.5	0.409	Unrolled-IG	37	27912	216.64	623.9	0.576

(e) LOCUS-AEAD						(f) SKINNY-AEAD					
Implementation	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (μ W)	Energy (nJ/128-bit)	Implementation	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (μ W)	Energy (nJ/128-bit)
1-Round	1036	5969	265.39	84.96	1.048	1-Round	560	8011	493.32	159.1	1.079
1-Round-CG	1036	5724	287.34	55.6	0.686	1-Round-CG	560	7451	400.22	136.3	0.924
2-Round	518	6471	402.89	102.6	0.634	2-Round	280	8701	645.88	200.9	0.683
2-Round-CG	518	6229	503.15	80.1	0.495	2-Round-CG	280	8205	683.44	184.7	0.628
3-Round	370	7035	522.40	132.2	0.585	3-Round	190	11109	682.02	320.5	0.742
3-Round-CG	370	6688	540.54	102.4	0.453	3-Round-CG	190	10546	648.47	304.4	0.705
4-Round	259	7445	617.76	175.2	0.544	4-Round	140	12890	691.48	528.4	0.904
4-Round-CG	259	7048	597.03	171.9	0.524	4-Round-CG	140	12354	783.67	513.8	0.879
Unrolled	37	19410	819.99	3880.6	3.582	Unrolled	10	69155	1422.22	26581.4	7.588
Unrolled-IG	37	27455	216.58	615.6	0.568	Unrolled-IG	10	110012	829.85	2125.9	0.607

(g) Romulus						(h) ForkAE					
Implementation	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (μ W)	Energy (nJ/128-bit)	Implementation	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (μ W)	Energy (nJ/128-bit)
1-Round	514	5729	642.19	123.2	0.782	1-Round	752	7362	363.88	159.4	1.335
2-Round	262	6315	657.24	153.1	0.497	1-Round-CG	752	6841	330.87	135.9	1.138
3-Round	181	8960	665.06	286.4	0.644	2-Round	380	8433	487.39	198.7	0.843
4-Round	136	10398	662.80	472.9	0.801	2-Round-CG	380	7618	478.92	173.3	0.735
Unrolled	19	68095	751.79	26480.1	13.409	3-Round	251	9863	666.13	309.0	0.868
Unrolled-IG	19	79277	683.63	4513.2	2.285	3-Round-CG	251	9125	587.66	301.5	0.847
						4-Round	190	12082	485.44	548.7	1.170
						4-Round-CG	190	11608	482.74	528.0	1.126
						Unrolled	9	103713	1606.63	55318.4	13.418
						Unrolled-IG	9	166923	1177.01	15418.5	3.740

Table 5. Measurements for the Pyjamask and SATURNIN implementations.

(a) Pyjamask						(b) SATURNIN					
Implementation	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (μ W)	Energy (nJ/128-bit)	Implementation	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (μ W)	Energy (nJ/128-bit)
1-Round-CG	180	15158	1485.04	193.3	0.387	1-Round	273	15214	638.78	413.8	1.255
2-Round	96	19552	1956.26	467.1	0.498	1-Round-CG	273	14540	622.96	382.6	1.161
2-Round-CG	96	19184	1959.60	426.5	0.454	2-Round	143	20530	2226.89	564.8	0.897
3-Round	72	26707	2287.67	897.4	0.718	2-Round-CG	143	19184	2226.89	531.3	0.844
3-Round-CG	72	26353	2287.67	859.0	0.687	4-Round	78	22895	2062.23	858.1	0.744
4-Round	60	34363	2249.45	1354.9	0.903	4-Round-CG	78	22160	2092.87	823.3	0.714
4-Round-CG	60	34031	2241.19	1315.1	0.876	Unrolled	13	70348	3322.58	37791.1	5.459
Unrolled	12	60540	4027.83	8602.7	1.147	Unrolled-IG	13	87854	2491.91	4790.6	0.623
Unrolled-IG	12	65610	2491.91	2494.1	0.333						

by a register bank in between, which suppresses the glitches produced by the TI of G . The approach has been summarized in Fig. 6a.

The structure has an additional disadvantage that they require 2 clock cycles to compute the shared s-box output, whereas a 4-share implementation would require only one cycle. Since time efficiency is also an equally important component of energy efficiency, this implies that it is not immediately evident that a 3-share would beat 4-share, as far as energy consumption of a TI is concerned. To make a fair evaluation of the energy efficiency of the AEAD schemes we implemented first order TI with the following characteristics:

1. We implemented first order TI of only round based circuits. This is necessary because r -round unrolled circuits must necessarily have higher algebraic degree, and as per the observation in [24], it will require more shares to construct a TI. For example a 2-round unrolled TI of a block cipher with a cubic s-box has algebraic degree 6, if properly designed and then 7 shares are required. The exact algebraic forms of each bit of a 7-share of a 1st order TI is likely to be very complicated, due to high degree, with multiple terms in each expression, eventually leading to large costs in area and power.
2. We implement TI profiles in which only the state path of the underlying encryption primitive is shared, but not the keypath. Many previous papers have taken this approach [11, 25], as it is adequate for first order security and for simplicity we follow the suit. If the keypaths were also shared, we estimate that it would increase the power consumption of the AEAD schemes by a similar factor, and energy consumption comparisons would probably lead to similar results. In short, we implement threshold circuits for all the AEAD schemes except SATURNIN. The mode SATURNIN was designed in an unusual way that the output of block cipher is used as the key in the subsequent block cipher call. And so a TI which only considers shares in the datapath is not possible for this mode.

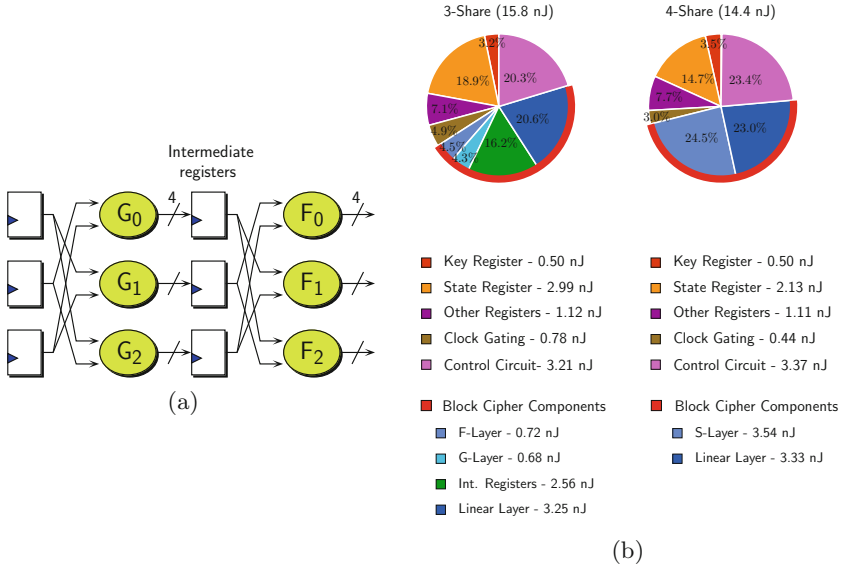


Fig. 6. (a) TI of a cubic s-box in 3 shares, (b) Energy consumptions in Pyjamask.

5.1 S-Box Details

Most of the schemes benchmarked in this work have cubic s-boxes with 4-bit inputs that are decomposable into quadratics $F \circ G$, and so efficient 3 and 4-share implementations are possible.

GIFT: The s-box of GIFT belongs to the cubic class \mathcal{C}_{172} which is decomposable into 2 quadratics using a direct sharing approach. The algebraic expressions of the output shares of both the 3 and 4-share TI can be found in [20].

Pyjamask-BC: The s-box of Pyjamask-BC belongs to the cubic class \mathcal{C}_{223} which is decomposable into 2 quadratics also using a direct sharing approach.

SKINNY: The s-box S_8 of SKINNY takes 8 input bits and has algebraic degree equal to 6. Therefore, a single cycle implementation would require 7 shares. Instead, we implement only a 3-share TI of all SKINNY based modes based on the recommendation given by the designers in [11]. S_8 can be decomposed into $\mathcal{I} \circ \mathcal{H} \circ \mathcal{G} \circ \mathcal{F}$ where each of these functions is an 8-bit quadratic s-box. In this particular case, a MUX is placed at the output of \mathcal{F} , \mathcal{G} , \mathcal{H} and \mathcal{I} (which is merged with the round function circuit). The algebraic expressions of the output shares of the 3-share TI can be found in [11, page 32].

5.2 Results

Table 6 lists the simulation results using the same measurement setup as the unshared round-based implementations (see Table 1). It can be seen that the schemes using SKINNY consume most energy, which is intuitive since the s-box

needs 4 clock cycles for evaluation. On the other hand, it is surprising to see that 4-share TI circuits have similar energy-efficiency when compared to the corresponding 3-share circuits.

Table 6. Measurements for the 1-round threshold implementations. The schemes using GIFT are colored in light gray whereas, SKINNY based schemes are in white

Candidate	Conf.	Shares #	Latency (cycles)	Area (GE)	TP _{max} (Mbps)	Power (mW)	Energy (nJ/128-bit)
GIFT-COFB	CG	3	800	16386	208.9	0.214	2.243
	CG	4	400	25850	350.8	0.358	1.875
SUNDAE-GIFT	-	3	1440	13297	145.7	0.215	3.719
	-	4	720	21848	285.2	0.357	2.999
HYENA	CG	3	800	14769	344.9	0.212	2.216
	CG	4	400	24540	497.4	0.358	1.875
LOTUS-AEAD	CG	3	2072	14176	121.7	0.145	3.581
	CG	4	1036	19712	133.0	0.262	3.232
LOCUS-AEAD	CG	3	2072	12366	121.7	0.137	3.362
	CG	4	1036	17597	176.8	0.255	3.148
SKINNY-AEAD	CG	3	2240	18501	92.83	0.2264	6.134
Romulus	CG	3	2056	13450	130.00	0.1865	4.656
ForkAE	CG	3	3008	17008	76.60	0.2483	8.304
Pyjamask	CG	3	348	42001	620.2	0.472	1.825
	CG	4	180	64577	927.6	0.814	1.628

One of the reasons for the above observation can be justified as follows: the fact that a 3-share circuit takes 2 cycles to evaluate s-box works against it. To understand the reasons better, we re-ran the Pyjamask simulations (with `-no_automgroup` directive to the compiler) and obtained a breakdown of the energies consumed by individual circuit components to process 1 associated data and 8 plaintext blocks. A summary is presented in Fig. 6.

One can see that whereas the energy consumed by the other components are comparable, the shared s-box layer (marked as S-Layer in the figure) of the 4-share TI consumes a lot of energy which is to be expected because the shares are algebraically more complicated. However, the 3-share TI does one additional operation, which the 4-share TI is not required to do, and that is writing values output by the G-layer on to the intermediate register bank. As it turns out these intermediate register writes consumes almost as much energy as the shared s-box circuit in the 4-share TI. Thus on average the energy consumed by the block cipher components in both the implementations balance out.

6 Final Observations and Conclusion

We tried to give a comprehensive guide to designing energy-efficient authenticated encryption schemes by evaluating a selection of ten NIST LWC candidates

that make use of a lightweight or a semi-lightweight block cipher at their core. In the process we were able to look at each candidate individually and identify optimal circuit configurations that would reduce the energy consumption of the AEAD circuit as a whole. We were also able to make broader observations regarding energy efficiency in AEAD modes instantiated with lightweight block ciphers. In the second part of the paper, we turned our attention towards threshold implementations. We looked at both 3-share and 4-share threshold implementations of the schemes and made energy measurements. For schemes based on SKINNY, the fact that 4 cycles are required to evaluate the shared s-box, means that the AEAD scheme must sacrifice more of its energy for the cipher itself. For the other candidates we note an up to 20% decrease in energy consumption for the 4-share implementation in comparison to the 3-share designs. We conclude our paper with the following claims, which applies to block cipher based AEAD paradigm, that can hopefully help achieve the ultimate goal of lightweightness with respect to energy consumption.

1. The size of register banks play an important role, in energy and area of an r -round unrolled AEAD circuit.³ In order to achieve efficiency, the designers should favor choices which lead to fewer number of storage elements, i.e. utilize a small number of temporary variables as possible in the mode of operation.
2. The r -round unrolled implementations strike a good balance between area, throughput and the energy consumption. However, the optimal value for r depends both on the block cipher and the surrounding mode of operation. Designers are recommended to experiment with different choices of r for the full AEAD scheme, and keep in mind that experiments based solely on block ciphers are not sufficient.
3. If a given AEAD scheme contains many storage elements, implementors are recommended to employ techniques such as clock-gating as much as possible to reduce the energy consumption. The efficiency of the clock-gating scales up with the number of idle storage elements and the total number of clock cycles during which they remain inactive.
4. From the energy perspective, there is almost a direct correlation between the lightweightness of non-threshold and threshold implementations of an AEAD scheme. Hence the optimal design choices for TI align well with the aforementioned decisions.

Acknowledgments. The 2nd and 3rd authors are supported by the Swiss National Science Foundation (SNSF) through the Ambizione Grant PZ00P2_179921.

References

1. NIST Lightweight Cryptography Project. <https://csrc.nist.gov/projects/light-weight-cryptography>

³ The Claim 1 is based on the fact that for 1-round unrolling of GIFT-COFB and SUNDABE-GIFT, more than half of the energy is consumed by the registers in Fig. 3, even though these two have relatively fewer flip-flops. On the other hand, percentage of energy consumption by registers are much higher for LOTUS-AEAD, because the mode of operation brings many intermediate variables into the circuit which need extra registers to store.

2. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Vizár, A.R.D.: Forkae vol 1. NIST Lightweight Cryptography Project (2019). <https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates>
3. Banik, S., Balli, F., Regazzoni, F., Vaudenay, S.: Swap and rotate: lightweight linear layers for SPN-based blockciphers. Cryptology ePrint Archive, Report 2019/1212 (2019). <https://eprint.iacr.org/2019/1212>
4. Banik, S., et al.: SUNDAE-GIFT v1.0. NIST Lightweight Cryptography Project (2019). <https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates>
5. Banik, S., Bogdanov, A., Regazzoni, F.: Exploring energy efficiency of lightweight block ciphers. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 178–194. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31301-6_10
6. Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-AES: a compact implementation of the AES encryption/decryption core. In: Dunkelman, O., Sanadhya, S.K. (eds.) INDOCRYPT 2016. LNCS, vol. 10095, pp. 173–190. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49890-4_10
7. Banik, S., Bogdanov, A., Regazzoni, F., Isobe, T., Hiwatari, H., Akishita, T.: Round gating for low energy block ciphers. In: 2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016, McLean, VA, USA, 3–5 May 2016, pp. 55–60 (2016). <https://doi.org/10.1109/HST.2016.7495556>
8. Banik, S., Bogdanov, A., Regazzoni, F., Isobe, T., Hiwatari, H., Akishita, T.: Inverse gating for low energy encryption. In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018, Washington, DC, USA, 30 April–4 May 2018, pp. 173–176 (2018). <https://doi.org/10.1109/HST.2018.8383909>
9. Banik, S., et al.: GIFT-COFB v1.0. NIST Lightweight Cryptography Project (2019). <https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates>
10. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Yu., Sim, S.M., Todo, Y.: GIFT: a small present. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 321–345. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_16
11. 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
12. Beierle, C., et al.: SKINNY-AEAD and SKINNY-Hash. NIST Lightweight Cryptography Project (2019). <https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates>
13. 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
14. Canteaut, A., et al.: Saturnin: a suite of lightweight symmetric algorithms for post-quantum security. NIST Lightweight Cryptography Project (2019). <https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates>
15. Chakraborti, A., Datta, N., Jha, A., Lopez, C.M., Nandi, M., Sasaki, Y.: Lotus-AEAD and Locus-AEAD. NIST Lightweight Cryptography Project (2019). <https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates>
16. Chakraborti, A., Datta, N., Jha, A., Nandi, M.: Hyena. NIST Lightweight Cryptography Project (2019). <https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates>

17. Goudarzi, D., et al.: Pyjamask v1.0. NIST Lightweight Cryptography Project (2019). <https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates>
18. Homsirikamol, E., et al.: CAESAR Hardware API. Cryptology ePrint Archive, Report 2016/626 (2016). <https://eprint.iacr.org/2016/626>
19. Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: Romulus v1.2. NIST Lightweight Cryptography Project (2019). <https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates>
20. Jati, A., Gupta, N., Chattopadhyay, A., Sanadhya, S.K., Chang, D.: Threshold implementations of GIFT: a trade-off analysis. *IEEE Trans. Inf. Forensics Secur.* **15**, 2110–2120 (2020). <https://doi.org/10.1109/TIFS.2019.2957974>
21. Jean, J., Moradi, A., Peyrin, T., Sasdrich, P.: Bit-sliding: a generic technique for bit-serial implementations of SPN-based primitives. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 687–707. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_33
22. Kerckhof, S., Durvaux, F., Hocquet, C., Bol, D., Standaert, F.-X.: Towards green cryptography: a comparison of lightweight ciphers from the energy viewpoint. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 390–407. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_23
23. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: a very compact and a threshold implementation of AES. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 69–88. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_6
24. Nikova, S., Rijmen, V., Schl affer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptol.* **24**(2), 292–321 (2010). <https://doi.org/10.1007/s00145-010-9085-7>
25. Poschmann, A., Moradi, A., Khoo, K., Lim, C.-W., Wang, H., Ling, S.: Side-channel resistant crypto for less than 2,300 GE. *J. Cryptol.* **24**(2), 322–345 (2010). <https://doi.org/10.1007/s00145-010-9086-6>



Cross-Site Search Attacks: Unauthorized Queries over Private Data

Bar Meyuhas¹(✉) , Nethanel Gelernter²(✉), and Amir Herzberg³(✉) 

¹ Bar-Ilan University, Ramat Gan, Israel
bar.meyuhas@gmail.com

² Cyberpion & College of Management Academic Studies, Rishon LeZion, Israel
nethanel.gelernter@gmail.com

³ University of Connecticut, Storrs, USA
amir.herzberg@gmail.com

Abstract. *Cross-site search attacks* allow a rogue website to expose private, sensitive user-information from web applications. The attacker exploits timing and other side channels to extract the information, using cleverly-designed cross-site queries.

In this work, we present a systematic approach to the study of cross-site search attacks. We begin with a comprehensive taxonomy, clarifying the relationships between different types of cross-site search attacks, as well as relationships to other attacks. We then present, analyze, and compare cross-site search attacks; We present new attacks that have improved efficiency and can circumvent browser defenses, and compare to already-published attacks. We developed and present a *reproducibility framework*, which allows study and evaluation of different cross-site attacks and defenses.

We also discuss *defenses* against cross-site search attacks, for both browsers and servers. We argue that server-based defenses are essential, including restricting cross-site search requests.

1 Introduction

When a user requests a page from a web-application, the response sent to the browser often includes instructions for the browser to make additional requests for other content. These additional requests are often cross-site requests to a different domain. Cross-site requests are allowed by many websites and widely used to include off-site content in a web page. Unfortunately, they are also abused for *cross-site attacks*.

Cross-site (XS) attacks are often launched by a malicious website that is accessed by the user, who is unaware of the risk. The malicious website asks the browser to send a request to a benign website that contains private information belonging to the client; the ultimate goal is to expose that information to the malicious website. Such information may be user data stored by the web-service (e.g., webmail) or private meta-data related to the user (e.g., information on the user's browsing history or previous interactions with the web-services). CSRF,

XSS, and Clickjacking [19,27] are common cross-site attacks. Cross-site attacks are easy to launch, as they do not require Man In The Middle (MITM) or even eavesdropping capabilities.

Defenses against such cross-site attacks are the cornerstone of browser security. The main defense implemented by all browsers is the *Same Origin Policy (SOP)*. SOP prevents scripts at one origin (site) from directly accessing content at a different origin (site).

Side channel attacks exploit attributes such as size and timing to obtain insight about the data, even though the access is restricted and the data are encrypted. It has been shown that side-channel information, such as packet timing, makes it possible to break cryptography systems or infer keystrokes in SSH [25]. For example, Zhang et al. [28] hijacked user accounts in a cloud environment using a cache-based side-channel.

Cross-site side-channel attacks are a combination of these two attack types. They can be used to bypass cross-site defenses using side-channel leaks. SOP may not be able to prevent the exposure of information via cross-site side-channels attacks. This particular attack type was demonstrated in several works. Heiderich et al. [18] tried to extract the cross-site request forgery (CSRF) token in order to bypass CSRF-defenses, while Bortz and Boneh [2] focused on extracting the current state of the web-page view. Malicious CAPTCHA [14] has also been used to trick the user into disclosing private information. In another attack, cross-site side-channel was used to detect whether a user is contacting a given web-site, even when the user uses Tor browser [16].

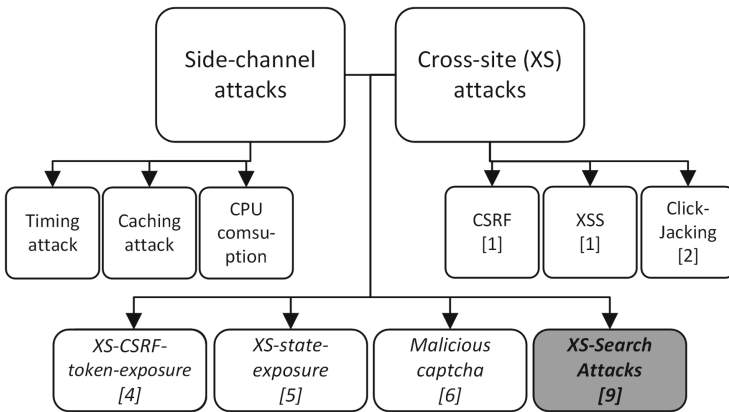


Fig. 1. Cross-site side-channel attacks are a combination of two attack types: cross-site and side-channel. This particular combination can be used to bypass cross-site defenses using side-channel leaks [2, 14, 18]. XS-Search attacks are also an example of this combination [13]. Similar to cross-site attacks, XS-Search attacks allow a script from a rogue website to perform an attack on data of other websites. Yet, similar to side-channel attacks, XS-Search attacks exploit side-channel analysis to expose the user’s private information.

In this paper, we investigate *cross-site search (XS-Search) attacks*. XS-Search attacks abuse the search capability provided by many web-services to expose the private information of a user. This may be user data stored in the site (e.g., for webmail) or private meta-data related to the user (e.g., information on the user’s browsing history). XS-Search attacks leverage the fact that when a search query has several matches, the resulting size of the response when there is a match will usually be larger than the size of a response that does not have any matches i.e., an *empty response*. Moreover, the computation time, and the transmission and processing times are often related to the number of matches in the response. XS-Search attacks are a composite of cross-site attacks and side-channel attacks; Fig. 1 presents the relationship between XS-Search attacks and other categories of cross-site and side-channel attacks. XS-Search attacks are considered *cross-site attacks* since they are launched by a rogue website and exploit requests sent from the browser, which includes the user’s cookie in some web-services, to expose private information related to the user and their operations in this web service. Because SOP restrictions prevent the rogue web-page from directly accessing the response received from the web-service, these attacks also employ *side-channel* techniques that provide information about the response-time and/or response-length.

Figure 2 depicts an example of a XS-Search attack that exploits the side-channel analysis of responses to queries on the victim’s account. This is done by checking if the victim’s name is a specific guess, say ‘victim’. The attacker sends a pair of queries to a web service like Gmail. The first is a challenge query for all messages sent by ‘victim’, which will obviously match many records. The second is a dummy query that will not match any records. We use the notation f_{SC} to denote a function that exposes the side-channel information. The attacker analyzes the information and tries to detect whether the challenge query returns ‘many’ records (i.e., user name is indeed ‘victim’) or no results (i.e., user name is not ‘victim’).

Similar to other side-channel attacks, XS-Search attacks face significant challenges. These include delays that depend on dynamically changing factors, such as congestion and concurrent processes in the client and server. Moreover, the side-channel leak is often influenced by several different factors, including client bandwidth, server load, client CPU performance, and more. Section 5 evaluates several methods to overcome these errors, such as inflation methods to increase the difference between empty and non-empty responses.

We investigate two XS-Search side-channels: length-based and time-based. Section 3 reviews several time-based XS-Search attacks, which use time measurements to distinguish between search-responses that contain results and responses that are empty with no results. Length-based XS-Search attacks use the *length* of the response as a side-channel. In Sect. 4, we introduce the first effective length-based XS-Search attack. This attack exploits browser features to discover the *exact* length of the response; this overcomes the imprecision of time measurements and exposes information more efficiently.

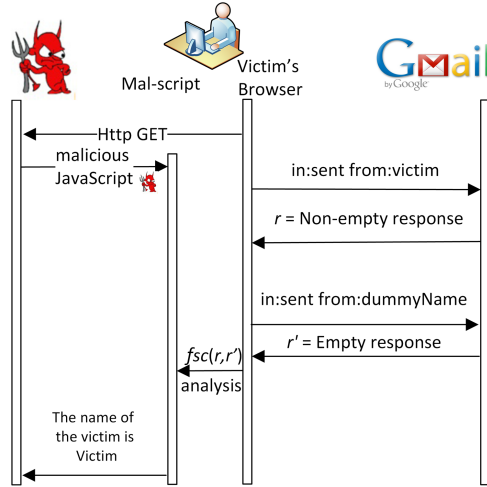


Fig. 2. An example of XS-Search attack exposing the name of the victim. The malicious script instructs the victim’s browser to send a pair of search queries to the web service (e.g., Gmail). The second search query has a ‘dummy’ query that will not match any records but serves as a baseline for the side-channel analysis. The malicious script analyzes the side-channel information exposed by f_{SC} . Time measurements and length samples can serve to distinguish between non-empty and empty responses.

Contributions of this work:

1. **Taxonomy, presentation and evaluation of XS-Search attacks.** XS-Search attacks were presented in [13, 15]; we provide a taxonomy, overview, and evaluation of these and other novel attacks, to provide a complete, organized view of this non-trivial threat vector. This includes a description of the primary known XS-Search attacks, including the Network-Time (NT) XS-Search, Cache-Time (CT) XS-Search, and Processing-Time (PT) XS-Search attacks described in Sects. 3.1, 3.2, and 3.3, respectively.
2. **Introduction of highly-efficient length-based (LB) XS-Search attack.** LB XS-Search, described in Sect. 4, exploits new browser features to directly and precisely discover the length of responses. This results in improved performance, compared to time-based XS-Search attacks as described in Table 1. We performed responsible disclosure of the vulnerability (and assigned CVE), and it was fixed in recent browser release.
3. **Reproducible evaluation of XS-Search attacks and defenses.** Previous works reported XS-Search results based on experiments conducted on ‘real’ web-services. Such results are interesting and we also perform this kind of evaluation. However, such experiments are not reproducible. Changes in the sites, their content, and even the network conditions can influence the results. To allow reproducible experiments, we set up an infrastructure that is independent of external web-services (available on GitHub [17]). This infras-

structure allows us, and other researchers, to investigate and evaluate known and new XS-Search attacks as described in Sect. 7.

4. **Defenses.** In Sect. 8, we present guidelines and defense methods to help web-services preclude XS-search attacks, including attacks that may exploit new browser vulnerabilities.

Table 1. This table presents the results of two experiments: extracting a credit card number from a Gmail email account and detecting whether the victim sent an email to a specific user (in the Enron dataset). See full experiment details in Sect. 7.

XS-Search attack	Gmail: credit card theft			Enron Dataset: expose contact from the sent emails		
	Success rate (%)	Average time (sec)	Number of requests	Success rate (%)	Average time (sec)	Number of requests
NT (Sect. 3.1)	90	1200	2000	82	110.9	2000
CT (Sect. 3.2)	96	60	115	95	45.32	2
LB (Sect. 4)	100	10	115	100	0.5	2

2 Taxonomy of XS-Search Attacks and Related Work

As mentioned earlier, XS-Search attacks are both cross-site attacks and side-channel attacks. In Sect. 1 we presented related works and attacks, Fig. 1 presents the relationship between XS-Search attacks, and other categories of cross-site and side-channel attacks.

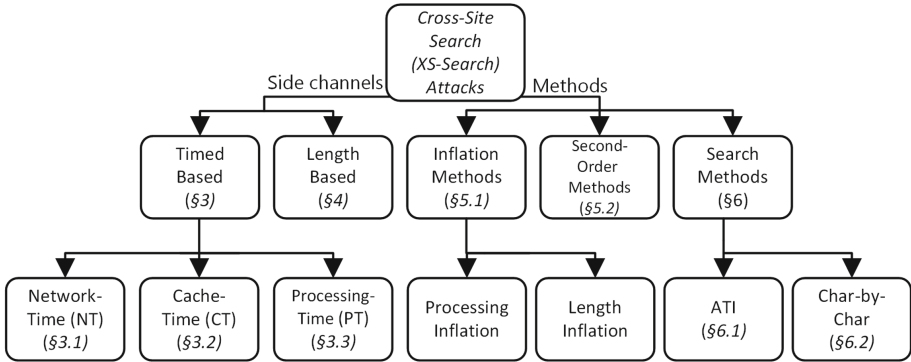


Fig. 3. XS-Search attacks exploit side-channel analysis to expose a user’s private information. There are two exploitable side-channels: length and time. This figure presents several attacks that exploit these side-channels and several methods to upgrade these Boolean attacks. These methods can be applied to allow efficient exposure of complex data.

In this section, with the aid of Fig. 3, we present the taxonomy of all XS-Search attacks and methods. In Subsect. 2.1, we describe two XS-Search attack types and in Subsect. 2.2, we present several methods to improve and upgrade these attacks.

2.1 XS-Search Attack Types

XS-Search attack performs side-channel analysis of responses to queries against the victim’s account. We investigate two XS-Search side-channels: length-based and time-based.

1. Timed-based XS-Search attacks [13, 15]:
 - 1.1 Network-Time (NT) XS-Search (see Sect. 3.1) attempts to distinguish between responses that produced search results and responses that did not, using numerous network time (round-trip delay) measurements.
 - 1.2 Cache-Time (CT) XS-Search (see Sect. 3.2) is based on measuring the time it took to load the response from the browser’s cache. This is as opposed to measuring the time the response loads from the network, as is done for NT XS-Search attacks.
 - 1.3 Processing-time (PT) XS-Search (see Sect. 3.3) is based on causing a significant difference in the time required to process the search queries in the server. This difference can often be significant for carefully engineered queries, such as queries that are constructed as a conjunction of many terms. This exploits the fact that these complex queries may cause excessive computation time in the server. The attack was shown to be effective only for simple Boolean queries.
2. Length-Based (LB) XS-Search (introduced in Sect. 4). While previous timed-based XS-Search attacks used time measurements to learn the length of a response, LB XS-Search exploits the browser’s features to discover it directly. These vulnerable browser features allow LB XS-Search to overcome the lack of determinism in previous attacks and more effectively expose the length of the cross-site response.

2.2 XS-Search Methods

XS-Search attacks face significant challenges, these include delays that depend on dynamically changing factors, such as congestion and concurrent processes in the client and server. We evaluate several methods to overcome these errors:

1. *Inflation* methods (see Subsect. 5.1) work by increasing the difference between non-empty and empty responses so they can be distinguished from one another.
 - 1.1 *Response-Length Inflation* increases the length of responses using inflating parameters in the query.
 - 1.2 *Processing-Time Inflation* increases the time it takes the server to process the responses, causing excessive computation time in the server using carefully engineered queries.

2. *Second-Order* (see Subsect. 5.2) methods use storage manipulation to take advantage of the victim. The attacker adds to the victim’s account (e.g., email inbox), multiple records that contain information in which the attacker is interested (e.g., reset password token). When the data appears in many records, the attacker can search for it and detect whether the response is empty or contains many records.
3. *Search methods* (see Subsect. 6) We show several search methods to efficiently resolve complex queries:
 - 3.1 *Any-Term Identification* (ATI) (see Subsect. 6.1) is a *term-identification* algorithm. Term-identification algorithms are useful when the attacker has a large set S containing n potential search terms, e.g., name of person/location/project/other names, phone-numbers, credit-card numbers, or passwords. The attacker wants to identify which, if any, of these terms appear in the private data records of the client.
 - 3.2 Char-by-Char (see Subsect. 6.2) can be used when the indexing technique in the attacked service is based on partial strings and the search interface supports searching based on parts of strings.

For the sake of simplicity, Sects. 3 and 4 focus on the ability to answer Boolean questions about the user. Sections 5 and 6 suggest several methods that can upgrade the simple Boolean attacks to expose complex data e.g., credit-card number.

3 Time-Based XS-Search Attacks

Time-based XS-Search attacks exploit the fact that the loading time of an HTTP response in the browser provides information about the response’s content. Specifically, the loading time of a non-empty and an empty response is often different. This difference could be due to the processing time of the request or due to the size of the response. The attack measures the time it takes to receive a response resulting from search requests.

To launch the attack, the attacker sends multiple pairs of challenge requests alongside dummy requests. Based on the response time values that are measured, the attacker can decide whether the challenge request resulted in an empty response or a non-empty response, i.e., there were some matching records. Using statistical tests on both measurements can indicate whether they were drawn from similar distributions. If they weren’t drawn from similar distributions, the challenge request resulted in a non-empty response (See Fig. 2).

3.1 Network-Time (NT) XS-Search Attack

NT XS-Search attacks focus on the network delays and measure the round-trip delay of search requests. The transmission and the loading time of a resource depends on its size: the larger the resource, the more time it takes to transmit and load it. The size of non-empty responses is greater than the size of empty

responses, thus resulting in a longer measured delay time. To overcome factors that affect the measurements, such as client bandwidth and server load, the attacker has to send each request multiple times. To improve the accuracy of the measurements, the attacker can also use inflation techniques. Inflation techniques increase the difference between non-empty and empty responses, making them more distinguishable (see Subsect. 5.1).

The Caching Challenge. To speed up web browsing, browsers are designed to download web pages and store them locally in the browser cache. When the same page is visited for a second time, the browser loads the page locally from the cache instead of downloading it from the network. This may foil the NT XS-Search attack in two ways. First, it may cause the ‘dummy’ query to be served directly from the cache. Second, if multiple queries are sent for the same request to improve precision using many samples, then all but the first query may be served from the cache, without any network delay or server-processing delay.

To overcome this challenge, the attacker concatenates a random dummy parameter to each request. This forces the browser to send the request to the server since it is not identical to the previous requests. As a result, the measurement is not affected by browser caching.

3.2 Cache-Time (CT) XS-Search Attack

CT XS-Search attack exploits the fact that responses loaded from the browser cache create a timing side-channel that allows information to be leaked. The duration of a measurement from the cache is significantly lower than a measurement from a remote server. Therefore, an attacker can compensate for the smaller difference in the loading time by taking many local measurements. An additional advantage of the CT XS-Search attack is that every request is sent only once to the server; this reduces the chance of detection by shortening the duration of the attack. A basic CT XS-Search attack tries to answer a Boolean question. The attack sends the server only two requests, challenge and dummy, and stores the responses in the browser cache. The attack then collects time measurements locally in the victim’s browser by loading the responses from the cache several times.

The Cache-Time (CT) XS-Search attack is more robust than the NT XS-Search attack because it is not affected by the delay between the victim and the server, nor by the variability in the processing delay at the server. An additional advantage of this method is the fact that every request is sent only once to the server, reducing the chance of possible detection.

3.3 Processing-Time (PT) XS-Search Attack

Processing-Time (PT) XS-Search attacks are based on causing a significant difference in the time required to process the search queries; this difference can often be significant for carefully engineered queries. The attack exploits the fact

that these queries may cause excessive computation time in the server. A PT XS-search attack is applicable when the following conditions are satisfied:

1. Multiple term queries - Queries can be constructed as a conjunction of two or more terms.
2. Early abort - Once any term(s) return false, the entire processing is aborted (returning false) without waiting to complete the computation of any remaining terms.

Consider a search query containing the conjunction $\sigma \wedge \theta$, where σ is an easily-computed search term and θ is a hard-to-compute term. With the ‘early abort’ condition, if there is no record matching σ , the service will immediately return a negative response (no matching record). Conversely, if there is a matching record and σ resolves to true, the service will proceed to evaluate the ‘hard’ term θ , and return results only after that evaluation is completed. This technique provides a timing side-channel that allows an attacker to determine whether or not the response contains records matching the σ term.

4 Length-Based XS-Search Attack

In this section, we present the LB XS-Search attack, illustrated in Fig. 4b. This attack is based on direct, precise measurements of the *length* of the responses. We first describe how the attack uses the length of a response to find the number of records returned in the response. Then, we present several methods that provide a precise measurement of the length of the cross-site response.

4.1 Computing the Number of Records from Response-Length

We now explain how the adversary can compute the number of records returned for a specific search query, based on the length of the response. The computation works for the common case where search responses have the structure shown in Fig. 4a. Namely, the response consists of a fixed length meta-data field, whose length we denote as ϕ , and of ρ bytes for each record returned.

The adversary first finds the values of the constants ϕ and ρ for the attacked search web-service. This is done once and can typically be done in advance before sending requests from the script to the service.

Assume, therefore, that ϕ and ρ are known. The malicious script sends a search query, assuming the script can learn the length l of the response, perhaps using one of the methods described in the following subsections. The number of records in the response is given by: $r = (l - \phi) / \rho$. The process is illustrated in Fig. 4b.

If the length of the responses exactly follows the structure of Fig. 4a, and the length l is found precisely, then a single query, or very few queries, suffice to know the exact number of records returned in a response. This allows quick and precise leakage of private information. See the results in Table 1.

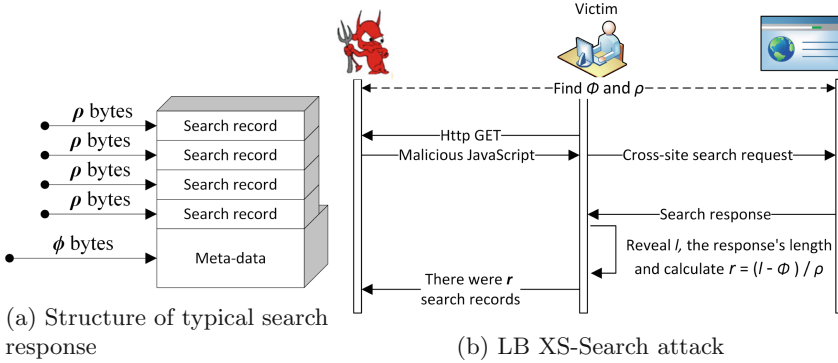


Fig. 4. Figure 4a depicts structure of typical search response allowing identification of the number of records returned, given the length of the response. Figure 4b depicts LB XS-Search attack, a malicious script sends a cross-site search query to a web service, whose responses are structured as in Fig. 4a. The number of records returned r is given by: $r = (l - \phi) / \rho$.

```

navigator.webkitTemporaryStorage.queryUsageAndQuota(
  function(usedBytes, grantedBytes) {
    console.log('we are using', usedBytes
               'of', grantedBytes, 'bytes');
  }

```

Listing 1.1. The `queryUsageAndQuota` function of the Chrome Quota API, returning the storage used by the cache. The function is used by Algorithm 1 to calculate the size of a response.

4.2 Measuring the Response Length

We found several methods that provide a precise measurement of the length of the cross-site response.

Chrome Quota Management API. This method to expose the length of the cross-site response exploits implementations of the Chrome *service workers*. Service workers allow HTTP responses to be saved in cache storage to give users a more efficient and pleasant experience. This component also enables developers to create and intercept HTTP requests and responses [23].

Chrome allows the developers of service workers to manage cache storage using the Quota Management API [3]. Part of this API is the `queryUsageAndQuota` function, which allows a script to check the remaining storage in the cache; it is called as in Listing 1.1. This function returns the storage size that is being used and the available space left. Using Algorithm 1 and the Chrome Quota API, we were able to calculate the exact size of the query response.

The vulnerability was reported in June 2016 [4] and was exposed in Google Chrome from Version 51 (May 2016) to Version 62 (Oct 2017).

Algorithm 1. Calculate a stored cross-site response size in a service worker’s cache using Chrome Quota API

```

1: // Sample the usage before storing in the cache
2: used_before ← queryUsageAndQuota()
3: // Store the response in the service worker’s cache
4: cache.put(response)
5: // Sample the usage after storing in the cache
6: used_after ← queryUsageAndQuota()
7: // Calculate the response size
8: return used_after - used_before

```

Chrome Padding Responses Mechanism. Subsection 4.2 explained how the `queryUsageAndQuota` function of the quota management API allows the cross-site exposure of response size in Versions 51 to 62 of the Chrome browser. To support the functionality provided by the Quota API without exposing the cross-site response size, a padding mechanism was developed and integrated into Chrome version 63. We next show how the attacker can circumvent this padding mechanism and still expose the response size.

Let us first explain the padding mechanism. In this mechanism, the saved response size is the sum of the actual response size plus an additional ‘padding’. The goal is for the amount of this padding to be random, i.e., unpredictable to the attacker. This way, the cache usage does not reflect the actual response size and the designers hoped to foil the exposure of the response length. Specifically, the padding size is calculated using a hash-based function (HMAC) on the request URL, using a secret key k known only to the browser. Simplifying a bit:

$$\text{Padding size} = \text{HMAC}_k(\text{requestURL}) \bmod \text{MAX_PAD} \quad (1)$$

The challenge for the attacker is to find the padding size for the victim domain, since the key k used to compute it is unknown. We show how the attacker can compute it. The solution is based on the observation that the padding size is the same for all responses from the same URL, as computed in Eq. (1).

Recall that the ultimate goal of the attacker is not to find the padding size or even the response size. The attacker’s goal is to find the number of records r in the response. We show how the attacker can still find r , using the following steps, illustrated in Fig. 5:

1. Send search request to the URL, *without cookies*. We used an advanced feature of the Fetch API that allows us to omit credentials from the sent request [12]. This results in a fixed-length response, since the attacker is not authenticated using the cookie as required. For simplicity, assume the response is of length ϕ , i.e., the length of the meta-data is included as part of the search response, as in Fig. 4a. Then, the attacker uses Algorithm 1, to find the length of the padded response:

$$x = \phi + (\text{HMAC}_k(\text{requestURL}) \bmod \text{MAX_PAD}).$$

- Attacker repeats the previous step, for the ‘real’ search request, to the same URL, *with cookies*. The goal of the attacker is to find the number of records r returned in this response, where each record is of known length ρ , as shown in Fig. 4a. Then, the attacker uses Algorithm 1 again to find the length of this padded response:

$$x' = \phi + r\rho + (HMAC_k(requestURL) \bmod MAX_PAD).$$

- The number of records in the ‘real’ response is $r = \frac{x' - x}{\rho}$.

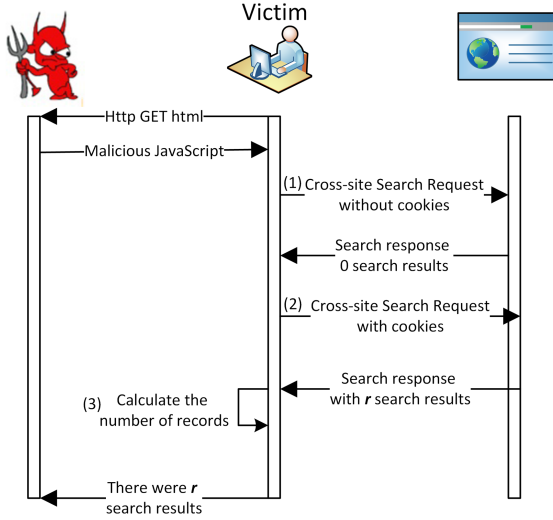


Fig. 5. LB XS-Search attack process using Chrome’s padding mechanism vulnerability. Malicious script sends two cross-site search requests to the *same URL*: the first *without cookies* and the second *with cookies*. The first results in a fixed-length response since the attacker is not authenticated as the victim. The second response holds r search records since the server authenticates the victim. The padding size of the two responses is the same as per Eq. 1. The number of records in the ‘real’ response is $r = \frac{x' - x}{\rho}$

The vulnerability was exposed in Google Chrome from Version 63 (Dec 2017) to Version 81 (March 2020). We disclosed this vulnerability [5], which was assigned CVE-2020-6442. Google addressed this vulnerability from Version 81 (March 2020), making it safe for us to present in this publication. To prevent the exposure from Version 81, Google added another element to the padding calculation. This element is a flag that indicates whether the request includes credentials. In this way, the browser uses different padding for responses to requests with and without credentials (cookies):

$$\text{Padding size} = HMAC_k(requestURL + credentials_flag) \bmod MAX_PAD \quad (2)$$

5 Optimizations

One of the core challenges in XS-Search is to distinguish between non-empty and empty responses. In this section, and in Appendix A of the full version of this paper [20], we present several optimizations that an attacker can perform to improve the attack’s performance.

5.1 Inflation Methods

Launching a naive timing attack, as described in Fig. 2, is challenging. In many cases, the difference between the responses is not significant enough to efficiently distinguish between them. *Inflation* is a technique that helps overcome this problem by increasing the difference between non-empty and empty responses. We consider two inflating techniques: *processing-time inflation* and *response-length inflation*.

Processing-time inflation directly increases the time it takes the server to process our queries. It causes an excessive amount of computation time at the server using carefully engineered queries (see Subsect. 3.3 for details).

Response-length inflation increases the length of the responses. This way, the time it takes to transmit and load responses may also increase. Response-length inflation is based on three conditions that hold for a parameter in the search request, which we refer to as the *inflated parameter*:

1. It is possible to send long strings, with possibly thousands of characters, as the value of the inflated parameter.
2. The inflated parameter appears only a few times, or not at all, in an empty response.
3. The inflated parameter appears many times in a non-empty response, usually as a function of the number of entries in the response.

By sending a relatively long inflated parameter, the length of the response is significantly influenced by the number of entries. This often allows the attacker to distinguish between empty responses with no matches versus responses with a significant number of matches. This optimization can be applied with all types of XS-Search attacks. (See more details and examples in [13].)

5.2 Second-Order (SO) Optimization

The NT XS-Search and CT XS-Search attacks discussed in Sects. 3.1 and 3.2, respectively, are limited to pieces of information that appear many times in the search responses. This makes it crucial to have a noticeable difference between the size of the responses that contain records with the searched terms and those that are empty.

Second-Order (SO) optimization is a technique that *creates* such a difference between the sizes of the responses, even when only a single record matches the challenge search request. This optimization manipulates the victim’s account

(e.g., email inbox) by *adding* multiple records that contain the information in which the attacker is interested. Once the data appears in many records, the attacker can search for it and detect whether the response is empty or contains many records.

The challenge of the attacker is to inject records holding sensitive information into the account of the victim, without knowing the content of that information. For example, the attacker can cause third-party websites to send emails to the victim’s account and then steal the information from these emails. A common example is to ask third-party web services to reset the victim’s password using a reset password email. From such reset password emails, the attacker can extract a password-reset token (e.g., in Facebook), or details that the victim entered in the third-party website (e.g., username). See more details and examples in [15].

6 Term-Identification Query Algorithms

The attacks presented in the previous sections answer a single Boolean query. In this section, we show how to efficiently resolve *term-identification queries*. In a term-identification query, the attacker has a large set S containing n potential search terms, e.g., person/location/project/other names, phone-numbers, credit-card numbers, or passwords. The attacker wants to identify which, if any, of these terms appear in the private data records of the client. The solutions use divide and conquer to identify the relevant term in S , using a series of Boolean queries.

A naïve term-identification method will perform n single term queries. Because n is often large (e.g., the number of possible names), this attack may require that the victim remains connected to the rogue site for an unreasonably-long period of time. Therefore, this naïve method may have limited value in practice.

In this section we present two term-identification algorithms: the *Any-Term Identification* algorithm in Subsect. 6.1, and the Char-by-Char Search algorithm in Subsect. 6.2. Due to length restrictions, we present the Optimized-Any-Term-Identification algorithm in Appendix C of the full version of this paper [20].

6.1 Any-Term Identification (ATI) Algorithm

The ATI algorithm can find one term from the set S that appears in the private data. The attacker assumes that at least one of the terms appears in the private data. Assuming set S contains at least one term that has search results, simplifying the algorithm’s goal to find any value in S for which the search query has results, allows ATI to avoid the use of dummy queries. Instead, ATI divides the set S into two subsets: S_1 and S_2 . The algorithm compares the responses from the two subsets and continues the search with the greater response set, based on time or length analysis. The ATI continues recursively until one element remains.

6.2 Char-by-Char Search Algorithm

During our study, we investigated dozens of vulnerable search interfaces and examined the different indexing techniques used by websites to handle their data (see Table 2 in the full version of this paper [20]). Most of the search interfaces we investigated used the standard method of index by string, specifically whole strings. Nonetheless, some of the web-services, even vastly popular ones, used a more flexible and advanced indexing method that allows indexing and search using *partial strings*.

The *Char-by-Char Search algorithm* can be used when the indexing technique is based on partial strings, and the search interface supports searching based on parts of strings.

The attack proceeds by searching for all available chars, one at a time. The query with the most results contains the first char of the desired information. Afterward, we search for the second char together with the char we already found. This proceeds until the complete string is revealed.

Char-by-Char Search is an efficient search algorithm, but it is not necessarily an optimal one. In Appendix C of the full version of this paper [20], we present some algorithmic improvements and optimizations to further decrease the number of sent requests.

7 Experiments

In this section, we present several experiments conducted to compare the performance and results of all XS-Search attacks. We also introduce our reproducibility system, which allows the reproduction of our experiments, attacks, and defenses. Due to length restrictions, more experiments, details (network conditions, operating system, browser version, etc.) and a list of the most popular vulnerable services that we exploit and data we were able to extract are available in Appendix D and E of the full version of this paper [20]. We performed responsible disclosure and allowed vendors to address the vulnerability before publication.

7.1 Reproducibility

Web services are very dynamic and frequently updated; some of these updates may impact the site’s vulnerability to different attacks, including changes in response to disclosures (such as our disclosure). Furthermore, XS-Search attacks depend on the content stored by the web-service for a particular user, as well as unpredictable network and processing delays, which may change over time. This makes it difficult to reproduce results and to compare different methods fairly and precisely.

To ensure the reproducibility of our experiments despite these challenges, we set up an infrastructure that is independent of external web-services. This infrastructure was built on a virtual machine that embodies all the technologies necessary to perform XS-Search attacks:

1. Local mail service that allows cross-site search requests, and supports simple and complex queries. To simulate real user mailboxes, we used the Enron dataset, which contains data from about 150 users, mostly senior management of the Enron corporation [10].
2. Deter-Lab service, which allows us to simulate real network conditions, including packet-losses and delay, in a reproducible manner [9].
3. Web service that presents the results of the search requests in a user-friendly interface [11].
4. Service that simulates cross-site attacks and allows performing XS-Search attacks. We implemented three XS-Search attacks: NT, CT, and LB. (See experiment details in Subsect. 7.2.)

Using this infrastructure, researchers can reproduce our experiments and analyze the results. We hope this infrastructure will help others investigate, implement, and perform more XS-Search attacks. It should also allow researchers to compare their results to the results of existing attacks. We made our attack scripts, reproducible system, and some other information available on GitHub [17].

7.2 The Reproducible Enron XS-Search Experiment

The reproducible Enron XS-Search experiment is part of our reproducibility environment. We used the Enron dataset to simulate 150 email users and performed our attacks against this simulated mail-service. The Enron dataset consists of more than 200,000 emails, mostly from the senior management of Enron. The dataset is unique in that it is one of the only publicly available mass collections of actual emails that has been made easily available for study.

Attack Goal: Map all Enron employees and detect the leaders of the company. We wrote a malicious script that checked whether an individual employee sent emails to the VP of Enron. We assumed that the leaders of the company sent emails to the VP of the company, and if they did not then they were not part of the high-level management. An attacker who wants to map the leaders of the Enron company can use that script and send it to all the employees he would like to examine. We implemented three scripts that executed this scenario; each script launched a specific type of XS-Search attack: NT, CT, and LB described in Sects. 3.1, 3.2, and 4, respectively.

Our experiment attempted to identify 5 Enron employees who were email users; we repeated the test 10 times for each user. Table 1 presents the results of the experiment. LB achieved the best results with 100% success. CT achieved the second-best results, with a lower success rate. The NT attack was far inferior to the other two, not only in terms of success rate and average time, but also in the number of requests sent to the server. Although both LB and CT required only 2 requests, the NT attack required 2000 (!) requests. As mentioned in Subsect. 3.2, the CT attack collects time measurements locally (instead of sending many requests to the server) in the victim’s browser by loading the responses from the cache several times.

7.3 Gmail: Credit Card Number Experiment

Gmail is the world’s most popular mail service and is used by more than 1.4 billion [24] active users. It supplies an advanced mailbox with progressive features for email correspondence. Gmail allows its clients to conduct search queries and complex filtering for data in their mailboxes.

Inside the mailbox, Gmail supplies several search interfaces that allow hackers to expose a credit card number (16 digits) via search. We launched an attack in January 2019 using LB XS-Search to expose a credit card number from a victim’s mailbox. As opposed to CT XS-Search and NT XS-Search in which the credit card number had to appear in several distinct emails (at least five), LB XS-Search succeeded in revealing a 16-digit credit card number that appeared *only once*.

Credit card numbers have a fixed structure, such as dddd-dddd-dddd-dddd. Even if we assume the card number is completely random, every 4-digit group has 10,000 possible values. The special structure of credit card numbers allows us to use term identification algorithms (e.g., ATI) to efficiently extract individual card numbers.

We compared several XS-Search attacks in Table 1. The timed-based XS-Search attacks sent many requests and measured them several times in order to achieve a higher success rate. NT was launched using length-inflation and ATI. CT was launched using length-inflation, second-order, and OATI. LB XS-Search supplied determinism and higher accuracy, even though each request was measured and sent only once. LB was launched using OATI only. Thanks to the single measurement policy, the attack’s duration was reduced from 1200 s (for the NT attack) or 60 s (for the CT attack), to less than 10 s.

8 Defense Techniques

8.1 Client-Side Defenses

Hardening Quota Management API. To prevent more Length-Based attacks and block the attack method mentioned in Subsect. 4.2, a padding mechanism was developed by Google to store cross-site responses in the cache; this way the saved size does not reflect the original response size. Unfortunately, the mechanism developed had an additional vulnerability (described in Subsect. 4.2). Van Goethem, et al. suggest [26] a mechanism to pad the responses and store them in the cache. Appropriate programming of this mechanism could have prevented the attack technique described in Subsect. 4.2.

Cross Origin Read Blocking (CORB). Cross Origin Read Blocking (CORB) has been a feature in Chrome since version 67 [7], and was developed to prevent dubious cross-site requests. The feature uses an algorithm that identifies abnormal requests and replaces their responses with empty ones. For example, a request from an IMG tag whose response is an HTML page would return

empty. The mechanism uses the response headers, such as Content-Type, to decide whether or not a request should be blocked. As of today, CORB restricts the use of cross-site search queries and prevents some of the attacks presented in this paper. Nonetheless, CORB does not provide a foolproof solution. The algorithm decides whether a request is suspicious according to a predetermined set of rules, which are updated occasionally [8]. Moreover, the latest version of CORB does not filter cross-site requests sent by extensions and external players (such as Flash Player) [6]. It is possible that, in the future, a vulnerability will be found that allows sending cross-site search requests.

8.2 Server-Side Defenses

Anti-CSRF. The term Cross-Site Request Forgery (CSRF) is usually applied to describe attacks that either commit ‘an unwanted action’ or affect the server state. Although XS-Search attacks do not commit an unwanted action, they do expose private data using *abusive cross-site requests*. Known anti-CSRF techniques can be applied to block these abusive requests. These include challenge-response (CAPTCHA), anti CSRF-tokens, referrer verification, and same-site cookies [1, 21, 22]. Applying any of these anti-CSRF mechanisms to *all* requests that may expose sensitive data, directly or indirectly, would prevent any type of XS-Search attacks. The web relies on connectivity. Sites load resources such as images, advertisements, scripts, and other elements from different sites. As a result, it is impossible to completely block all cross-site requests.

Rate Limiting. In the past, restricting the number of requests sent to the server prevented XS-Search attacks. Rate limiting is especially effective against time-based XS-Search attacks because these attacks require numerous requests to leak information effectively. That said, rate limiting is futile against LB XS-Search attacks because the attack does not depend on time measurements and hence does not require many requests. Moreover, the attacks in Sect. 3.2 rely on browser-based timing side channels; therefore, adding a delay in the server will not be effective. It is also critical to block the inflation techniques. The inflating SO XS-Search attacks presented in Sect. 5.2 are based on the injection of maliciously crafted records that significantly increase the size of the response. Blocking such records, will prevent these particular attacks.

9 Conclusions

With the growing improvements and adoption of web-security mechanisms such as encrypted-connections and sanitation against cross-site scripting, attackers are moving to new and more sophisticated attacks. In this paper, we discuss a new, sophisticated attack vector: cross-site search (XS-Search) attacks.

XS-Search attacks leverage the fact that when a search query has several matches, the resulting response size will usually be larger than it would be with an empty response. Using side-channel analysis, the attacker tries to detect

whether the query returns a sufficient number of results. We consider two side-channels: length and time.

This paper provides a taxonomy of XS-Search attacks and several methods to improve them. We describe several XS-Search attacks: Network-Time (round-trip delay), Cache-Time (loading from cache delay), Processing-Time (server process delay), and Length-Based (direct exposure using browser vulnerabilities).

We also compare these XS-Search attacks and defenses, evaluate them using several experiments. To ensure the reproducibility of our experiments, we set up a reproducibility system. It allows the reproduction of our experiments, attacks, and defenses. Our evaluation shows that XS-Search attacks successfully exposed sensitive information (e.g., credit card number) from popular web services such as Gmail, Facebook, and YouTube. (See Table 2 in the full version of this paper [20].)

References

1. Barth, A., Jackson, C., Mitchell, J.C.: Robust defenses for cross-site request forgery. In: Proceedings of the Conference on Computer and Communications Security (2008)
2. Bortz, A., Boneh, D.: Exposing private information by timing web applications. In: Proceedings of the 16th International Conference on World Wide Web, pp. 621–628. ACM (2007)
3. Managing Storage. https://developer.chrome.com/apps/offline_storage#managing_quota
4. Issue 617963: Security: Service workers response size info leak. <https://bugs.chromium.org/p/chromium/issues/detail?id=617963>
5. Issue - Chromium. <https://bugs.chromium.org/p/chromium/issues/detail?id=1013906>
6. Changes to cross-origin requests in chrome extension content scripts. <https://www.chromium.org/Home/chromium-security/extension-content-script-fetches>
7. Cross-origin read blocking for web developers. <https://www.chromium.org/Home/chromium-security/corb-for-developers>
8. Cross-Origin Read Blocking (CORB). https://chromium.googlesource.com/chromium/src/+master/services/network/cross_origin_read_blocking_explainer.md
9. DETERLab Capabilities. https://deter-project.org/deterlab_capabilities
10. Enron Email Dataset, May 2015. <https://www.cs.cmu.edu/~enron/>
11. Enron email frontend archive, October 2017. <https://github.com/antiboredom/enron-email-archive>
12. Fetch API standards. <https://fetch.spec.whatwg.org/#concept-request-credentials-mode>
13. Gelernter, N., Herzberg, A.: Cross-site search attacks. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1394–1405. ACM (2015)
14. Gelernter, N., Herzberg, A.: Tell me about yourself: the malicious captcha attack. In: Proceedings of the 25th International Conference on World Wide Web, pp. 999–1008. International World Wide Web Conferences Steering Committee (2016)

15. Gerlenter, N.: Advanced cross-site search attacks. https://owasp.org/www-pdf-archive//AppSecIL2016_AdvancedCrossSiteSearch_NethanelGelernter.pdf
16. Gilad, Y., Herzberg, A.: Spying in the dark: TCP and tor traffic analysis. In: Fischer-Hübner, S., Wright, M. (eds.) PETS 2012. LNCS, vol. 7384, pp. 100–119. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31680-7_6
17. GitHub: XS-Search Attacks. <https://github.com/barmey/xs-search>
18. Heiderich, M., Niemietz, M., Schuster, F., Holz, T., Schwenk, J.: Scriptless attacks: stealing the pie without touching the sill. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 760–771. ACM (2012)
19. Huang, L.S., Moshchuk, A., Wang, H.J., Schechter, S., Jackson, C.: Clickjacking: attacks and defenses. In: Presented as Part of the 21st {USENIX} Security Symposium, {USENIX} Security 2012, pp. 413–428 (2012)
20. Meyuhas, B., Herzberg, A., Gelernter, N.: Cross-site search attacks: unauthorized queries over private data (extended version), October 2020. https://www.researchgate.net/publication/344503497_Cross-Site_Search_Attacks_Unauthorized_Queries_over_Private_Data
21. OWASP: OWASP/CSRFCSheet, June 2019. <https://github.com/OWASP/CheatSheetSeries>
22. Same-site cookies RFC, April 2016. <https://tools.ietf.org/html/draft-west-first-party-cookies-07>
23. Service Workers: An Introduction. <https://developers.google.com/web/fundamentals/primers/service-workers/>
24. Smith, C.: 20 Amazing Gmail Statistics, June 2019. <https://expandedramblings.com/index.php/gmail-statistics/>
25. Song, D.: Timing analysis of keystrokes and SSH timing attacks. In: Proceedings of 10th USENIX Security Symposium (2001)
26. Van Goethem, T., Vanhoef, M., Piessens, F., Joosen, W.: Request and conquer: exposing cross-origin resource size. In: 25th {USENIX} Security Symposium, {USENIX} Security 2016, pp. 447–462 (2016)
27. Zalewski, M.: The Tangled Web: A Guide to Securing Modern Web Applications. No Starch Press, San Francisco (2012)
28. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-tenant side-channel attacks in PaaS clouds. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 990–1003 (2014)

Cybersecurity



Stronger Targeted Poisoning Attacks Against Malware Detection

Shintaro Narisada¹(✉), Shoichiro Sasaki², Seira Hidano¹,
Toshihiro Uchibayashi³, Takuo Suganuma⁴, Masahiro Hiji⁵,
and Shinsaku Kiyomoto¹

¹ KDDI Research, Inc., Fujimino, Japan

{sh-narisada, se-hidano, kiyomoto}@kddi-research.jp

² Graduate School of Information Sciences, Tohoku University, Sendai, Japan

sasaki.sh1234@ci.cc.tohoku.ac.jp

³ Research Institute for Information Technology, Kyushu University, Fukuoka, Japan

uchibayashi.toshihiro.143@m.kyushu-u.ac.jp

⁴ Cyberscience Center, Tohoku University, Sendai, Japan

suganuma@tohoku.ac.jp

⁵ Graduate School of Economics and Management, Tohoku University, Sendai, Japan

hiji@tohoku.ac.jp

Abstract. Attacks on machine learning systems such as malware detectors and recommendation systems are becoming a major threat. *Data poisoning attacks* are the primary method used; they inject a small amount of poisoning points into a training set of the machine learning model, aiming to degrade the overall accuracy of the model. *Targeted data poisoning* is a variant of data poisoning attacks that injects malicious data into the model to cause a misclassification of the *targeted* input data while keeping almost the same overall accuracy as the unpoisoned model. Sasaki *et al.* first applied targeted data poisoning to malware detection and proposed an algorithm to generate poisoning points to misclassify targeted malware as goodwill. Their algorithm achieved 85% an attack success rate by adding 15% poisoning points for malware dataset with continuous variables while restricting the increase in the test error on nontargeted data to at most 10%. In this paper, we consider common defensive methods called *data sanitization defenses*, against targeted data poisoning and propose a defense-aware attack algorithm. Moreover, we propose a stronger targeted poisoning algorithm based on the theoretical analysis of the optimal attack strategy proposed by Steinhardt *et al.* The computational cost of our algorithm is much less than that of existing targeted poisoning algorithms. As a result, our new algorithm achieves a 91% attack success rate for malware dataset with continuous variables by adding the same 15% poisoning points and is approximately 10^3 times faster in terms of the computational time needed to generate poison data than Sasaki's algorithm.

Keywords: Targeted data poisoning attack · Malware detection · Data sanitization

1 Introduction

Enterprises that hold sensitive data, such as personal information, are always at risk of leakage due to malware intrusion. To prevent attacks, malware detection technology plays an important role. Malware detection is traditionally based on pattern matching, which detects predefined anomaly patterns [26, 28]. Machine learning-based detection [1, 14, 15, 37] is considered to be more effective currently for high-rate attacks as typified by zero-day exploits. Deep learning-based detection can reduce the cost required to extract the complex and large features of malware and improve detection performance [17, 19, 31, 46].

The security risks of machine learning-based systems have been widely researched [5]. *Evasion attacks* (also called *adversarial examples*) assume that an attacker does not have permission to access the training data and that the attacker aims to fool the model by adding manipulated input. In recent work, Kolosnjaji *et al.* proposed an evasion attacks against a deep learning-based malware detection system that input is raw binary file [23]. *Data poisoning attacks* are another major attacks against machine learning systems. In data poisoning attacks, we assume that the attacker is able to inject the manipulated data into the training data. The purpose of the attack is to decrease the overall classification accuracy of the model. Biggio *et al.* [6] showed the effectiveness of this attack against the support vector machine (SVM). Newsome *et al.* [33] applied poisoning attacks to existing malware detection algorithms and confirmed that their proposed attacks are highly effective for malware detectors. Data poisoning attacks against deep learning algorithms have also been studied [32].

Since data poisoning attacks are indiscriminate attacks, an operator can easily detect the occurrence of the attack by checking the overall accuracy of the model. In contrast, *targeted data poisoning* is a *targeted* variant of data poisoning. In this attack, an attacker injects poison data into the training data so that only the *targeted* class is misclassified as the attacker’s desired class while the overall accuracy remains high. Targeted data poisoning is considered a more realistic variant of data poisoning attacks since the operator has difficulty noticing the attack. Shafahi *et al.* [40] proposed a poison generator of targeted data poisoning for the image classification problem. Sasaki *et al.* [38] presented a targeted data poisoning algorithm against malware detectors that uses a margin-based learning model (SVM and logistic regression).

1.1 Related Work

Backdoor Attacks. *Backdoor attacks* are variants of targeted data poisoning attacks. In this attack, an attacker injects poisoning samples into the training set that have a certain *trigger* added to them (*e.g.*, white noise or a transparent pattern for image classification). The goal of the attack is to have the input samples with the trigger misclassified into the attacker’s desired class. Chen *et al.* [9] proposed an attack against the DNN-based face recognition system, in which the trigger is wearing accessories. Liu *et al.* [30] applied backdoor attacks against speech recognition systems. Recently, Dai *et al.* [11] considered a

backdoor attack against the LSTM-based text classification problem. Targeted poisoning attacks against malware detection systems can be seen as a kind of backdoor attacks if a specific malware family is considered as the target, since a perturbation is embedded to the targeted data as a backdoor trigger to cause the misclassification of the targeted malware.

Defensive Methods for Poisoning Attacks. There are many studies on countermeasures against these attacks. Paudice *et al.* [34] proposed a defense mechanism against label flipping poisoning, where attackers are constrained to manipulate only labels. Baracaldo [4] presented a methodology to exclude online poisoned data from the training data by evaluating the provenance information of the data. Steinhardt *et al.* [42] computed a dataset-dependent upper bound on the maximum test loss that a poisoning attack causes when a defender uses *data sanitization defenses*, which remove the poisoned data by detecting statistical outliers [10]. For backdoor attacks, some research has focused on the information of the higher layers of the DNN model [29, 43]. There is also a way to identify a backdoor trigger using the reverse-engineering technique [44].

1.2 Contributions

While there are many proposals on defensive methods against nontargeted poisoning attacks, there has been little research about defensive methods for targeted poisoning attacks. In this paper, we apply the sphere defense (also called *L2 defense*) against targeted poisoning attacks to evaluate the performance of the defense for malware detection proposed in [38]. The sphere defense is known as an essential method to disable poisoning attacks mentioned in [22]. Our evaluation confirmed that the sphere defense indeed sanitizes existing targeted poisoning attacks.

Then, we propose stronger targeted poisoning algorithms that can evade the sphere defense: (1) a basic variant of the attacks that solves the optimization problem for targeted poisoning attacks naively, *i.e.*, by creating the poisoned data that minimize the validation loss; (2) a streamlined version of the basic variant, which is an online learning algorithm combines the label-flip attacks [45] and an idea from the optimal loss analysis proposed in [42]. We evaluate the effectiveness of these algorithms on multiple malware datasets. The experimental results show that both of proposed algorithms succeed against the attack even though the defender applies the sphere defense. Moreover, our algorithms achieved a 91% attack success rate for dataset with continuous variables for 15% poisoning, which exceeded the previous result of 85%. In terms of computation time, the streamlined algorithm is almost 1000 times faster than the basic gradient based algorithm. Our proposed algorithms enable a more efficient poisoning attack than existing methods in terms of the success rate and computational cost.

2 Preliminaries

2.1 Notation

We consider a binary classification from an input $\mathbf{x} \in \mathcal{X}$ to an output $y \in \mathcal{Y} = \{-1, +1\}$. The goal of the binary classification is to learn an objective function parameterized by \mathbf{w} that maps $\mathcal{X} \mapsto \mathcal{Y}$. Let ℓ be the loss function. In this paper, we use logistic regression for the model, and ℓ is logistic loss. We denote the training data set as $\mathcal{D}_{\text{train}}$ and the validation set as \mathcal{D}_{val} . The loss L for \mathcal{D} and \mathbf{w} is defined as $L(\mathbf{w}; \mathcal{D}) := \sum_{(\mathbf{x}, y) \in \mathcal{D}} \ell(\mathbf{w}; \mathbf{x}, y)$. Let \mathcal{M} be a learning algorithm.

2.2 Targeted Data Poisoning Attacks

We assume the same situation as (nontargeted) poisoning attacks, *i.e.*, an attacker has knowledge of the input space \mathcal{X} and learning algorithm \mathcal{M} . The attacker is able to read the training data $\mathcal{D}_{\text{train}}$ and inject the poisoned data \mathcal{D}_{p} into the training data, but he cannot remove any data from $\mathcal{D}_{\text{train}}$. The attacker chooses a target class and a desired class. The mean of “class” here depends on the problem to be solved. In malware detection problems, there are two classes: *malware* and *goodware*, and the target class is a *subclass* of the malware class, which consists of specific targeted malware.

The attacker’s objective is to construct a model that misclassifies points in target class $\mathbf{x}_{\text{target}}$ to desired class y_{desired} while keeping the accuracy of the other classes high. In binary classification, the desired class y_{desired} is the complement of the ground label of the targeted class, *i.e.*, \bar{y}_{target} . In this paper, we assume that any poisoning point \mathbf{x} has the desired label \bar{y}_{target} ; *i.e.*, the original label y_{target} is flipped to \bar{y}_{target} . One of the security applications of this attack is malware detection systems using federated learning [24] to learn malicious behavior from users’ applications. In federated learning, a model \mathcal{M} is trained in a collaborative manner among a number of participants. Thus \mathcal{M} is shared by all the participants, including an attacker. After that, an attacker can inject poisoning data \mathcal{D}_{p} into (a part of) training data $\mathcal{D}_{\text{train}}$. Then, a local model \mathcal{M} is trained from $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{p}}$. Finally, the poisoned model \mathcal{M} is uploaded to a central server.

A poisoning point $(\mathbf{x}, \bar{y}_{\text{target}}) \in \mathcal{D}_{\text{p}}$ is generated by solving the following bilevel optimization problem [32]:

$$\min_{\mathbf{x}} L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}}) \text{ s.t. } \hat{\mathbf{w}} \in \arg \min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}_{\text{train}} \cup \{\mathbf{x}, \bar{y}_{\text{target}}\}). \quad (1)$$

$\bar{\mathcal{D}}_{\text{val}}$ is the attacker’s validation data, where $\bar{\mathcal{D}}_{\text{val}} = \{(\mathbf{x}, \bar{y}_{\text{target}}) \mid (\mathbf{x}, y_{\text{target}}) \in \mathcal{D}_{\text{val}}\}$. The generated \mathcal{D}_{p} is injected into the clean training data $\mathcal{D}_{\text{train}}$. Then the learning algorithm trains on the combined training data $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{p}}$ and outputs a model parameter $\hat{\mathbf{w}}$. The attack is considered successful if $\hat{\mathbf{w}}$ classifies the targeted inputs as y_{desired} with a high probability and classifies nontargeted inputs accurately. We introduce several representative targeted data poisoning algorithms.

Algorithm 1: Muñoz’s Targeted Poisoning Algorithm [32]

Input: $\mathcal{D}_{\text{train}}, \bar{\mathcal{D}}_{\text{val}}, L$, the initial poisoning point $\mathbf{x}^{(0)}$, its flipped label \bar{y} , the learning rate η , a small positive constant ε

Output: the final poisoning point $\mathbf{x}^{(i)}$

- 1 $i \leftarrow 0, \text{loss} \leftarrow 0$
- 2 **do**
- 3 $\text{pre_loss} \leftarrow \text{loss}$
- 4 $\hat{\mathbf{w}} \in \arg \min_{\mathbf{w}} L(\mathbf{w}; (\mathcal{D}_{\text{train}} \cup \{\mathbf{x}^{(i)}, \bar{y}\}))$
- 5 $\mathbf{x}^{(i+1)} \leftarrow \mathbf{x}^{(i)} - \eta \nabla_{\mathbf{x}^{(i)}} L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}})$
- 6 $i \leftarrow i + 1$
- 7 $\text{loss} \leftarrow L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}})$
- 8 **while** $\text{pre_loss} - \text{loss} \geq \varepsilon$
- 9 **return** $\mathbf{x}^{(i)}$

Muñoz’s Algorithm [32]. Muñoz proposed an algorithm to solve Eq. 1 approximately and generate a poisoning point for targeted poisoning attacks. The pseudocode is given in Algorithm 1. In Line 4, the parameter $\hat{\mathbf{w}}$ is learned by a gradient descent algorithm so that $L(\mathbf{w}; \mathcal{D}_{\text{train}})$ decreases (from the defender’s perspective). The poisoning point $\mathbf{x}^{(i+1)}$ is updated by adding a gradient so that both the loss of $\mathcal{D}_{\text{train}}$ and $(\mathbf{x}^{(i)}, \bar{y})$ decrease (from the attacker’s perspective). In Line 5, we must compute the gradient of the loss for attacker’s validation data $\bar{\mathcal{D}}_{\text{val}}$ w.r.t. the current point $\mathbf{x}^{(i)}$. In [32], a back-gradient descent algorithm is applied to compute the gradient efficiently for deep learning models. Another solution is to use the influence function proposed in [21],

$$\nabla_{\mathbf{x}^{(i)}} L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}}) = -\nabla_{\hat{\mathbf{w}}} L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}}) H_{\hat{\mathbf{w}}}^{-1} \nabla_{\hat{\mathbf{w}}} \nabla_{\mathbf{x}^{(i)}} \ell(\hat{\mathbf{w}}; \mathbf{x}^{(i)}, \bar{y}). \quad (2)$$

where, $H_{\hat{\mathbf{w}}}^{-1}$ is a inverse of the hessian matrix $H_{\hat{\mathbf{w}}}$:

$$H_{\hat{\mathbf{w}}} = \lambda I + \nabla_{\hat{\mathbf{w}}} \nabla_{\hat{\mathbf{w}}} L(\hat{\mathbf{w}}; (\mathcal{D}_{\text{train}} \cup \{\mathbf{x}^{(i)}, \bar{y}\})). \quad (3)$$

A poisoning point is generated by iterating through several loops until the loss remains unchanged. The whole poisoned dataset \mathcal{D}_{p} is obtained by applying Algorithm 1 $|\mathcal{D}_{\text{p}}|$ times.

Extension by Sasaki [38]. Sasaki extends the Algorithm 1 by considering a constraint term w.r.t L2-norm between a poisoning point $(\mathbf{x}, \bar{y}_{\text{target}})$ and a randomly selected point in desired class $(\mathbf{x}_{\text{desired}}, y_{\text{desired}}) \in \mathcal{D}_{\text{train}}$ in order to avoid outlier detection by generating a poison data that has a label $\bar{y}_{\text{target}} = y_{\text{desired}}$ close to the distribution of the desired class. Namely,

$$\beta \|\mathbf{x} - \mathbf{x}_{\text{desired}}\|^2. \quad (4)$$

where, β is a hyperparameter that determines the effect of the constraint for $\beta \geq 0$. Eq. 4 is combined to the optimization formula (Eq. 1):

$$\begin{aligned} \min_{\mathbf{x}} \{L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}}) + \beta \|\mathbf{x} - \mathbf{x}_{\text{desired}}\|^2\} \\ \text{s.t. } \hat{\mathbf{w}} \in \arg \min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}_{\text{train}} \cup \{\mathbf{x}, \bar{y}_{\text{target}}\}). \end{aligned} \quad (5)$$

Algorithm 1 is modified in Line 5 to consider Eq. 5:

$$\mathbf{x}^{(i+1)} \leftarrow \mathbf{x}^{(i)} - \eta \nabla_{\mathbf{x}^{(i)}} L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}}) - \eta \beta \nabla_{\mathbf{x}^{(i)}} \|\mathbf{x}^{(i)} - \mathbf{x}_{\text{desired}}\|^2. \quad (6)$$

They evaluated the effectiveness of the algorithm varying β in the range of 0 to 0.2 against the L2-defense sanitizer. Their result showed the trade-off between the attack success rate and the detection rate, *i.e.*, the detection rate by the sanitizer decreases as β increases while attack success rate increases as β decreases. Therefore, targeted poisoning attack cannot keep the high attack success rate when a defender uses a sanitizer such as the L2-defense.

Shafahi’s Algorithm [40]. Shafahi presented optimization based targeted poisoning algorithm for image classification problem on neural networks. Let $f(\mathbf{x})$ be a function that propagates input \mathbf{x} through the neural network to the input of the softmax layer. Their optimization formula to craft a poisoning data is:

$$\min_{\mathbf{x}} \{\|f(\mathbf{x}) - f(\mathbf{x}_{\text{desired}})\|^2 + \beta \|\mathbf{x} - \mathbf{x}_{\text{target}}\|^2\}. \quad (7)$$

The algorithm to solve Eq. 7 approximately is shown in Algorithm 2. Algorithm 2 reduce the left and right term in Eq. 7 alternately.

At a high level, Eq. 7 and Eq. 5 in Sasaki’s extension are comparable. $\|f(\mathbf{x}) - f(\mathbf{x}_{\text{desired}})\|^2$ in Eq. 7 can be seen as a loss term between an output of the classifier from a poisoning point \mathbf{x} and an output from a data of desired class $\mathbf{x}_{\text{desired}}$. If the loss is small, then \mathbf{x} is expected to be classified as y_{desired} . Since $\bar{\mathcal{D}}_{\text{val}} = \{(\mathbf{x}, y_{\text{desired}}) \mid (\mathbf{x}, y_{\text{target}}) \in \mathcal{D}_{\text{val}}\}$ in Eq. 5, the left term of Eq. 7 (resp. Eq. 5) can be seen as a loss minimization between \mathbf{x} and $\mathbf{x}_{\text{desired}}$.

As for the right term in Eq. 5 and Eq. 7, $\beta \|\mathbf{x} - \mathbf{x}_{\text{target}}\|^2$ in Eq. 5 means crafting poisoning data \mathbf{x} close to $\mathbf{x}_{\text{desired}}$ intuitively. On the other hand, $\beta \|\mathbf{x} - \mathbf{x}_{\text{target}}\|^2$ in Eq. 7 yields \mathbf{x} near the targeted data $\mathbf{x}_{\text{target}}$. A unique feature of Shafahi’s attack is that they have succeeded in a clean label attack, in which poisoning data is crafted *without* flipping its label, by applying the transfer learning and watermarking techniques, while both Muñoz’s and Sasaki’s algorithm need label-flipping. In this paper, we consider only label-flipping attacks since as far as we experimented, no attacks succeeded without label-flipping in malware and benign software dataset.

We note that all of these three methods do not consider a defender applies a data sanitization to remove the poison data before a learning model \mathcal{M} trains the dataset $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{p}}$, or once the sanitization is applied, the attacks may become ineffective. It is more realistic to consider an anomaly detector to remove outlier from the training data that is implemented as a preprocessor in the learning phase.

Algorithm 2: Shafahi’s Targeted Poisoning Algorithm [40]

Input: initial poisoning point $\mathbf{x}^{(0)}$, desired poisoning point $\mathbf{x}_{\text{desired}}$, targeted poisoning point $\mathbf{x}_{\text{target}}$, learning rate η , constraint parameter β , iteration number N

Output: final poisoning point $\mathbf{x}^{(i)}$

```

1  $i \leftarrow 0$ 
2 for  $i < N$  do
3    $\hat{\mathbf{x}}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla_{\mathbf{x}^{(i)}} \|f(\mathbf{x}^{(i)}) - f(\mathbf{x}_{\text{desired}})\|^2$ 
4    $\mathbf{x}^{(i+1)} = \frac{\hat{\mathbf{x}}^{(i+1)} + \eta\beta\mathbf{x}_{\text{target}}}{1 + \eta\beta}$ 
5    $i \leftarrow i + 1$ 
6 return  $\mathbf{x}^{(i)}$ 

```

2.3 Data Sanitization Defense

Data sanitization defense [10] is the general term for the defenses that are employed in machine learning algorithms to detect anomalies in the training data by using statistical information from the training data. The *sphere defense* and *slab defense* are considered in [42] as instantiations of data sanitization defenses. The sphere defense removes points that are outside a spherical radius, and the slab defense projects points onto a line between the centroids of the classes, and eliminates points that are not on the line. For these defenses, the defender considers a feasible set $\mathcal{F} \subseteq \mathcal{X} \times \mathcal{Y}$ and \mathcal{M} trains only points on \mathcal{F} .

$$\hat{\mathbf{w}} := \arg \min_{\mathbf{w}} L(\mathbf{w}; (\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{p}}) \cap \mathcal{F}) \quad (8)$$

Let $\boldsymbol{\mu}_{-} := \mathbb{E}[\mathbf{x} \mid y = -1]$ and $\boldsymbol{\mu}_{+} := \mathbb{E}[\mathbf{x} \mid y = +1]$ be the centroids of the positive and negative classes. Then, the feasible set of defenses are defined as follows:

$$\mathcal{F}_{\text{sphere}} := \{(\mathbf{x}, y) : \|\mathbf{x} - \boldsymbol{\mu}_y\|_2 \leq r_y\} \quad (9)$$

$$\mathcal{F}_{\text{slab}} := \{(\mathbf{x}, y) : |\langle \mathbf{x} - \boldsymbol{\mu}_y, \boldsymbol{\mu}_y - \boldsymbol{\mu}_{-y} \rangle| \leq s_y\}, \quad (10)$$

where r_y and s_y are thresholds. In what follows, we consider only the sphere defense. In the field of anomaly detection research, the sphere defense is also called as a *centroid anomaly detection* and is widely used in anomaly-based intrusion detection systems for its low computational cost [7, 18, 27, 35]. Anomaly detection using *centroid-based clustering*, such as k -means, is an application of centroid anomaly detection techniques [3, 20]. The reason why we choose the sphere defense for the sanitizer is because of its efficiency as an anomaly detector and has many applications.

Steinhardt *et al.* [42] proposed an optimal attack strategy for (nontargeted) data poisoning attacks against the sphere defense and slab defense based on a worst-case analysis of the loss function. In Sect. 3.2, we apply this optimal attack strategy to the *targeted* poisoning attacks and propose an algorithm to efficiently generate poison in targeted attacks.

3 Sphere Defense-Aware Targeted Poisoning Attacks

For targeted poisoning attacks against malware detectors, we consider a binary classification problem in which a malware detector classifies an input x as either malware or goodware (benign software). As we mentioned in preliminaries, we use logistic regression for the malware detector. Some existing results show that neural networks and decision tree models outperform naive logistic regression algorithm in terms of detection accuracy [16, 41]. However, logistic regression is still useful for a malware detector when it is used in combination with other tools or algorithms [8, 25]. We consider the feasible set $\mathcal{F}_{\text{good}}$ of goodware for the sphere defense:

$$\mathcal{F}_{\text{good}} := \{(\mathbf{x}, y_{\text{good}}) : \|\mathbf{x} - \boldsymbol{\mu}_{y_{\text{good}}}\|_2 \leq r_{y_{\text{good}}}\}, \quad (11)$$

where, $\boldsymbol{\mu}_{y_{\text{good}}} := \mathbb{E}[\mathbf{x} \mid y = y_{\text{good}}]$ and $r_{y_{\text{good}}}$ is the threshold. In what follows, we assume that $r_{y_{\text{good}}}$ and $\boldsymbol{\mu}_{y_{\text{good}}}$ are pre-determined by a defender using the clean training data, *i.e.*, $r_{y_{\text{good}}}$ and $\boldsymbol{\mu}_{y_{\text{good}}}$ are determined by $(\mathbf{x}, y_{\text{good}}) \in \mathcal{D}_{\text{train}}$ for simplicity. Then, we consider an attacker-favorable situation in which the attacker notices that the defender is using the sphere defense as an anomaly detector and the attacker also knows $\mathcal{F}_{\text{good}}$ (both $\boldsymbol{\mu}_{y_{\text{good}}}$ and $r_{y_{\text{good}}}$). The sphere defense removes poison that is located in quite different place than clean data. Solving the loss-maximization problem yields poison located far away from clean data as is confirmed in the results for poisoning attacks to SVM [6]. Therefore, the sphere defense may be also effective against poisoning attacks in malware detection. The verification results of the sphere defense are shown in Sect. 4.4.

The attacker generates the poisoning data \mathcal{D}_{p} carefully so that each point is inside the sphere $\mathcal{F}_{\text{good}}$; *i.e.*, he cannot generate a point that will be considered as an outlier that has very high loss values. Then, the learning algorithm trains on $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{p}}$ by removing points outside the sphere. Note that no point in \mathcal{D}_{p} can be removed by the defense since the attacker generated the poisoning data so that they would not be removed.

The optimization problem to generate poison in Eq. 1 is rewritten as follows in the case of the malware detection problem with sphere defense:

$$\min_{\mathbf{x}} L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}}) \text{ s.t. } \hat{\mathbf{w}} \in \arg \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}; (\mathcal{D}_{\text{train}} \cup \{\mathbf{x}, y_{\text{good}}\}) \cap \mathcal{F}_{\text{good}}). \quad (12)$$

3.1 Basic Attack

We present a naive algorithm to generate poison by solving Eq. 12. The pseudocode is given in Algorithm 3. The main differences between Algorithm 3 and Algorithm 1 are as follows:

1. In Algorithm 3, the initial poisoning point x_0 is chosen randomly from the targeted malware in $\mathcal{D}_{\text{train}}$ so that (x_0, y_{good}) is located in $\mathcal{F}_{\text{good}}$,

$$\mathbf{x}_0 \stackrel{\S}{\leftarrow} \{\mathbf{x}_{\text{mal.B}} \mid (\mathbf{x}_{\text{mal.B}}, y_{\text{mal.B}}) \in \mathcal{D}_{\text{train}}, (\mathbf{x}_{\text{mal.B}}, y_{\text{good}}) \in \mathcal{F}_{\text{good}}\}. \quad (13)$$

where, mal.B corresponds to the label of the targeted malware.

Algorithm 3: Poison Generator against Sphere Defense

Input: $\mathcal{D}_{\text{train}}, \bar{\mathcal{D}}_{\text{val}}, L$, the initial poisoning point $\mathbf{x}^{(0)}$, its flipped label y_{good} , the learning rate η , a small positive constant ε

Output: the final poisoning point $\mathbf{x}^{(i)}$

```

1  $i \leftarrow 0$ 
2  $loss \leftarrow 0$ 
3 do
4    $pre\_loss \leftarrow loss$ 
5    $\hat{\mathbf{w}} \in \arg \min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}_{\text{train}} \cup \{\mathbf{x}^{(i)}, y_{\text{good}}\})$ 
6    $\mathbf{x}^{(i+1)} \leftarrow \mathbf{x}^{(i)} - \eta \nabla_{\mathbf{x}^{(i)}} L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}})$ 
7    $\mathbf{t} \leftarrow \boldsymbol{\mu}_{y_{\text{good}}} - \mathbf{x}^{(i+1)}$ 
8   if  $\|\mathbf{t}\| > r_{y_{\text{good}}}$  then
9      $\boldsymbol{\delta} \leftarrow \frac{\|\mathbf{t}\| - r_{y_{\text{good}}}}{\|\mathbf{t}\|} \mathbf{t}$ 
10     $\mathbf{x}^{(i+1)} \leftarrow \mathbf{x}^{(i+1)} + \boldsymbol{\delta}$ 
11    $i \leftarrow i + 1$ 
12    $loss \leftarrow L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}})$ 
13 while  $pre\_loss - loss \geq \varepsilon$ 
14 return  $\mathbf{x}^{(i)}$ 

```

- We check whether $\mathbf{x}^{(i+1)}$ is inside $\mathcal{F}_{\text{good}}$ each time the point is updated. If $(\mathbf{x}^{(i+1)}, y_{\text{good}}) \notin \mathcal{F}_{\text{good}}$, a perturbation $\boldsymbol{\delta}$ is added to $\mathbf{x}^{(i+1)}$ to project the point onto the sphere, *i.e.*, $(\mathbf{x}^{(i+1)} + \boldsymbol{\delta}, y_{\text{good}}) \in \mathcal{F}_{\text{good}}$, where $(\mathbf{x}^{(i+1)} + \boldsymbol{\delta}, y_{\text{good}})$ is closest to $(\mathbf{x}^{(i+1)}, y_{\text{good}})$ in terms of L2 distance.

The concept of Algorithm 3 is similar to the *influence attack* proposed in [22]. However, the two algorithms differ in the following ways: (1) the influence attack is a variant of the poisoning attack algorithm that aims to degrade the overall accuracy of the model, whereas Algorithm 3 is for targeted poisoning attacks. (2) Algorithm 3 updates the point $\mathbf{x}^{(i)}$ to decrease the loss of $\mathcal{D}_{\text{train}} \cup \{\mathbf{x}^{(i)}, y_{\text{good}}\}$, while the influence attack updates the point to increase it.

3.2 Streamlined Attack Based on the Optimal Attack Strategy

In this section, we study the worst-case loss analysis for targeted poisoning attacks. Then, we propose a targeted poisoning attack algorithm based on the analysis. We repost the optimization problem for targeted poisoning attacks (we omit the sphere defense term for simplicity).

$$\min_{\mathcal{D}_{\text{p}}} L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}}) \text{ s.t. } \hat{\mathbf{w}} \in \arg \min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{p}}). \quad (14)$$

In previously proposed algorithms [32, 42], a single poisoning point is computed by solving the optimization problem of minimizing the loss of \mathbf{w} multiple times. Algorithm 3 is also of this type. The problem for these algorithms is the high

computational cost for minimizing the loss of \mathbf{w} . To address this issue, we consider swapping the inner problem and outer problem in Eq. 14. Note that the technique of swapping the min-max problem was previously applied to normal poisoning attacks in [42]. In the following, we develop the discussion in a quite different way than for the method of nontargeted poisoning attacks.

First, we divide $\mathcal{D}_{\text{train}}$ into two distinct sets $\mathcal{D}_{\text{train}}^{\text{B}}$ and $\mathcal{D}_{\text{train}}^{\text{N}}$ depending on whether the points represent targeted malware: $\mathcal{D}_{\text{train}}^{\text{B}} = \{(\mathbf{x}_{\text{mal.B}}, y_{\text{mal.B}}) \in \mathcal{D}_{\text{train}}\}$ and $\mathcal{D}_{\text{train}}^{\text{N}} = \mathcal{D}_{\text{train}} \setminus \mathcal{D}_{\text{train}}^{\text{B}}$. Then, the validation loss $\bar{\mathcal{D}}_{\text{val}}$ can be replaced by the training loss $\mathcal{D}_{\text{train}}^{\text{B}}$ assuming that the validation data $\bar{\mathcal{D}}_{\text{val}}$ has approximately the same distribution as the training data $\mathcal{D}_{\text{train}}^{\text{B}}$.

$$\min_{\mathcal{D}_{\text{p}}} L(\hat{\mathbf{w}}; \bar{\mathcal{D}}_{\text{val}}) \approx \max_{\mathcal{D}_{\text{p}}} L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}^{\text{B}}) \quad (15)$$

Note that $\min_{\mathbf{x}} \ell(\mathbf{w}; \mathbf{x}, \bar{y})$ is equivalent to $\max_{\mathbf{x}} \ell(\mathbf{w}; \mathbf{x}, y)$ in the case of binary classification. We assume a set of poisoning data \mathbb{D}_{p} such that adding any element of \mathbb{D}_{p} to $\mathcal{D}_{\text{train}}$ does not noticeably affect the loss of nontargeted malware. Namely, $\mathbb{D}_{\text{p}} = \{\mathcal{D}_{\text{p}} \mid L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}^{\text{N}}) \approx L(\mathbf{w}_{\text{c}}; \mathcal{D}_{\text{train}}^{\text{N}})\}$, where, $\mathbf{w}_{\text{c}} \in \arg \min L(\mathbf{w}; \mathcal{D}_{\text{train}})$. This situation is quite natural for targeted poisoning because the training loss for all data except for the target can be kept as low as the loss of the clean trained model.

We show that if the poison data \mathcal{D}_{p} is chosen from \mathbb{D}_{p} , $\arg \max_{\mathcal{D}_{\text{p}}} L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}^{\text{B}})$ can be approximated by $\arg \max_{\mathcal{D}_{\text{p}}} L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}})$.

Proposition 1. *If $\mathcal{D}_{\text{p}} \in \mathbb{D}_{\text{p}}$, then,*

$$\begin{aligned} \arg \max_{\mathcal{D}_{\text{p}}} L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}^{\text{B}}) &\approx \arg \max_{\mathcal{D}_{\text{p}}} L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}) \\ \text{where, } \hat{\mathbf{w}} &\in \arg \min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{p}}). \end{aligned} \quad (16)$$

Proof. By the definition of \mathbb{D}_{p} , $L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}^{\text{N}}) \approx L(\mathbf{w}_{\text{c}}; \mathcal{D}_{\text{train}}^{\text{N}})$. Then,

$$\arg \max_{\mathcal{D}_{\text{p}}} L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}^{\text{B}}) \approx \arg \max_{\mathcal{D}_{\text{p}}} (L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}^{\text{B}}) + L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}^{\text{N}})). \quad (17)$$

Since $\mathcal{D}_{\text{train}} = \mathcal{D}_{\text{train}}^{\text{N}} \cup \mathcal{D}_{\text{train}}^{\text{B}}$ and $\mathcal{D}_{\text{train}}^{\text{N}} \cap \mathcal{D}_{\text{train}}^{\text{B}} = \emptyset$, we get the proposition. \square

Proposition 1 allows us to compute the maximum loss for the whole training set $\mathcal{D}_{\text{train}}$ instead of $\mathcal{D}_{\text{train}}^{\text{B}}$. Then, we show that maximal loss for $\mathcal{D}_{\text{train}}$ is bounded by the following proposition:

Proposition 2. *If $\mathcal{D}_{\text{p}} \in \mathbb{D}_{\text{p}}$, then,*

$$\max_{\mathcal{D}_{\text{p}}} L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}) \leq \min_{\mathbf{w}} \max_{\mathcal{D}_{\text{p}}} L(\mathbf{w}; \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{p}}). \quad (18)$$

Proof. $\max_{\mathcal{D}_p} L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}})$ is bounded by the loss of the training data plus the poisoning data:

$$\begin{aligned} \max_{\mathcal{D}_p} L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}) &\leq \max_{\mathcal{D}_p} (L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}}) + L(\hat{\mathbf{w}}; \mathcal{D}_p)) \\ &= \max_{\mathcal{D}_p} L(\hat{\mathbf{w}}; \mathcal{D}_{\text{train}} \cup \mathcal{D}_p), \\ &= \max_{\mathcal{D}_p} \min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}_{\text{train}} \cup \mathcal{D}_p). \end{aligned} \quad (19)$$

We apply the min-max theorem to swap the min and max and achieve the proposition. \square

Expanding the right term of Eq. 18 yields the following optimization problem:

$$\min_{\mathbf{w}} \left[L(\mathbf{w}; \mathcal{D}_{\text{train}}) + \max_{\mathcal{D}_p} L(\mathbf{w}; \mathcal{D}_p) \right]. \quad (20)$$

This equation suggests that we should construct a min-max swapped algorithm, which is similar to the poisoning attack algorithms proposed in [22, 42].

The remaining problem is that of computing $\mathcal{D}_p \in \mathbb{D}_p$. Our strategy is to plot the poisoning point at almost the same coordinates as the targeted points in $\mathcal{D}_{\text{train}}^B$ to increase only the loss $L(\mathbf{w}; \mathcal{D}_{\text{train}}^B)$. A basic strategy is to apply a variant of *label-flip attacks* [45]. In this attack, an attacker is allowed to create the poisoning point so that the initial point is copied from $\mathcal{D}_{\text{train}}^B$ and the label is flipped:

$$\mathcal{D}_p \subseteq \{(\mathbf{x}, \bar{y}) \mid (\mathbf{x}, y) \in \mathcal{D}_{\text{train}}^B\} \quad (21)$$

To include the intent of the loss maximization of Eq. 20 in Eq. 21, we consider maximizing the loss $\ell(\mathbf{w}; \mathbf{x} + \mathbf{d}, y)$, where \mathbf{x} is added to a perturbation vector \mathbf{d} restricted to $\|\mathbf{d}\| \leq r$ for a small radius r .

$$\mathcal{D}_p \subseteq \{(\mathbf{x} + \mathbf{d}, \bar{y}) \mid (\mathbf{x}, y) \in \mathcal{D}_{\text{train}}^B, \mathbf{d} \in \arg \max_{\mathbf{d}, \|\mathbf{d}\| \leq r} \ell(\mathbf{w}; \mathbf{x} + \mathbf{d}, \bar{y})\}. \quad (22)$$

We have no further theoretical support for the idea that the poison data generated by these equations must be in \mathbb{D}_p , but our evaluation shows that such a poisoning point almost satisfies $\mathcal{D}_p \in \mathbb{D}_p$ in practice.

We constructed an online learning algorithm that combines Eqs. 20, 21 and 22. Our proposed algorithm is shown in Algorithm 4. The parameter $\mathbf{w}^{(t)}$ is trained throughout the for loop by the gradient descent algorithm. Additionally, a single poisoning point $(\mathbf{x}^{(t)}, y^{(t)})$ is computed by solving the optimization for each loop. To evade the sphere defense, the initial point is chosen from $(\mathbf{x}, y) \in \mathcal{F}_{\text{good}}$, which is the same as in the maximization problem in Line 5. After *burn*+*n* loops, the algorithm outputs *n* poisoning points \mathcal{D}_p . There are some notable points in the algorithm:

Algorithm 4: Streamlined Poison Generator against Sphere Defense

Input: clean training data $\mathcal{D}_{\text{train}}$, feasible set $\mathcal{F}_{\text{good}}$, poisoning rate ε , learning rate η , burn-in number burn , optimization radius r

Output: \mathcal{D}_p

```

1  $\mathbf{w}^{(0)} \leftarrow 0$ 
2 for  $t \leftarrow 1, \dots, \text{burn} + n$  do
3   if  $t > \text{burn}$  then
4      $(\mathbf{x}^{(t)}, y^{(t)}) \stackrel{\$}{\leftarrow} \{(\mathbf{x}, \bar{y}) \mid (\mathbf{x}, y) \in \mathcal{D}_{\text{train}}^{\text{B}} \cap \mathcal{F}_{\text{good}}\}$ 
5      $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t)} + \underset{\mathbf{d}, \|\mathbf{d}\| \leq r, (\mathbf{x}^{(t)} + \mathbf{d}, y^{(t)}) \in \mathcal{F}_{\text{good}}}{\arg \max} \ell(\mathbf{w}^{(t-1)}; \mathbf{x}^{(t)} + \mathbf{d}, y^{(t)})$ 
6      $\mathcal{D}_p \leftarrow \mathcal{D}_p \cup \{(\mathbf{x}^{(t)}, y^{(t)})\}$ 
7      $\mathbf{g}^{(t)} \leftarrow \nabla L(\mathbf{w}^{(t-1)}; \mathcal{D}_{\text{train}}) + \nabla \ell(\mathbf{w}^{(t-1)}; \mathbf{x}^{(t)}, y^{(t)})$ 
8   else
9      $\mathbf{g}^{(t)} \leftarrow \nabla L(\mathbf{w}^{(t-1)}; \mathcal{D}_{\text{train}})$ 
10     $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \mathbf{g}^{(t)}$ 
11 return  $\mathcal{D}_p$ 

```

1. If $r = 0$, the algorithm generates poisoning points in Eq. 21: *i.e.*, it generates poison data by label-flip attacks.
2. To solve the maximization problem $\max_{\mathbf{d}, \|\mathbf{d}\| \leq r} \ell(\mathbf{w}; \mathbf{x} + \mathbf{d}, \bar{y})$, if the model is margin based, then maximizing $\ell(\mathbf{w}; \mathbf{x}, y)$ is equivalent to minimizing $y\mathbf{w}^T \mathbf{x}$. Since we use logistic regression as the model, maximizing $\ell(\mathbf{w}; \mathbf{x} + \mathbf{d}, \bar{y})$ is solvable by computing $\bar{y}\mathbf{w}^T (\mathbf{x} + \mathbf{d})$ with constraints $\|\mathbf{d}\| \leq r$ and $(\mathbf{x} + \mathbf{d}, \bar{y}) \in \mathcal{F}_{\text{good}}$. We use an optimization solver to compute it from the viewpoint of calculation efficiency.
3. We introduce a burn-in process in the for-loop to create a point only when $\mathbf{w}^{(t)} \approx \mathbf{w}_c$, in the same way as for the min-max poisoning attack in [22].

We evaluate the efficiency of the algorithm in the next section. The selection of an appropriate radius r and burn-in burn is also detailed in the next section.

4 Validation

4.1 Experimental Setup

We implemented all the algorithms in Python with PyTorch.¹ All experiments were conducted in Ubuntu 16.04 with two Intel Xeon E5-2697 3.3 GHz CPU. The memory consists of two 12 GB NVIDIA TITAN X VRAMs. We used two malware datasets Ransomware and M-EMBER, created by Sasaki *et al.* [38]. Ransomware is a dataset with discrete variables that is based on the dataset created in Sgandurra

¹ Our implementation is available on <https://github.com/mllearning-security/stronger-targeted-poisoning>.

et al. [39], in which the feature vector \mathbf{v} is binary and the dataset consists of 942 goodwillware and 582 ransomware. The dimensions of \mathbf{v} are reduced to 400 by using the mutual information shown in [39], namely, $\mathbf{v} \in \{0, 1\}^{400}$. M-EMBER is a dataset with continuous variables that is a combination of two datasets EMBER [2] and the Microsoft malware dataset [36]. M-EMBER includes 9 families of malware and 300,000 goodwillware. A feature vector of M-EMBER is the byte histogram of the file that represents the normalized value of the counts of each byte within the file. Namely, $\mathbf{v} \in [0, 1]^{256}$. We choose a malware as the target in Ransomware and M-EMBER as shown in Table 1. The remaining 8 malware families are used as nontargeted malware.

Table 1. The candidate of the targeted malware we used.

Dataset	Malware family	Malware ID
Ransomware	Critroni	1
Ransomware	MATSU	2
M-EMBER	Ramnit	3
M-EMBER	Tracur	4

The training data $\mathcal{D}_{\text{train}}$, the poisoning data \mathcal{D}_{p} , and the test data $\mathcal{D}_{\text{test}}$ are created as follows: $\mathcal{D}_{\text{train}}$ consists of 100 randomly selected goodwillware, 20 targeted malware and 80 nontargeted malware. The nontargeted malware consists of 8 malware families. We set the poisoning rate ε to $\varepsilon = |\mathcal{D}_{\text{p}}|/|\mathcal{D}_{\text{train}}|$. Our poison generator (Algorithms 3 and 4) constructs \mathcal{D}_{p} of size $\varepsilon|\mathcal{D}_{\text{train}}|$. We vary ε in the range $0 \leq \varepsilon \leq 0.15$ through validation. The clean test data $\mathcal{D}_{\text{test}}$ consists of 50 goodwillware, 15 targeted malware and 35 nontargeted malware, which are randomly selected. We used logistic regression for the malware detector. To solve the gradient of the loss for $\bar{\mathcal{D}}_{\text{val}}$ in Algorithm 1 and Algorithm 3, we apply the influence function method in Eq. 2. We set $\lambda = 10$ in Eq. 3. For the sphere defense, we set the threshold $r_{y_{\text{good}}}$ in Eq. 9 to remove 15% of the points from $\mathcal{D}_{\text{train}}$. We set $\beta = 0.01$ in Sasaki’s algorithm for ID1, ID2 and $\beta = 0.1$ for ID3, ID4. For the optimal radius r and the burn-in number $burn$ in Algorithm 4, we found that $r = 0.5$ and $burn = 2000$ are optimal for these malware families by searching with grid search. The learning rate η is 0.1. We additionally experimented with the label-flip variant of our algorithm (the case of $r = 0$). We refer to the case of $r = 0.5$ as **solver** and $r = 0$ as **flip** in what follows. We used the CVXPY Python-embedded solver [12] to compute the optimization problem of maximizing the loss for $\mathbf{x}^{(i)} + \mathbf{d}$. We averaged the results of 10 executions for validation. Shafahi’s targeted poisoning algorithm [40] is omitted in the validation since it is for image classification problems on a neural networks model.

Table 2. Poison removal ratio for each ID.

ID	Poison removal ratio			
	Algorithm 1	Sasaki’s algorithm	Algorithm 3	Algorithm 4
1	75.0%	0%	0%	0%
2	0.0%	0%	0%	0%
3	100.0%	5.6%	0%	0%
4	94.4%	2.8%	0%	0%

4.2 Evaluation Indicator

We evaluated the attack performance with the following indicators.

- *The attack success rate*, which measures how often targeted malware is classified as goodware in $\mathcal{D}_{\text{test}}$ (the false negative rate of the targeted malware).
- The rate at which the nontargeted malware is classified goodware in $\mathcal{D}_{\text{test}}$ (the false negative rate of the nontargeted malware).
- The rate at which the goodware is classified as malware in $\mathcal{D}_{\text{test}}$ (the false positive rate of the goodware).

4.3 Defensive Performance of Sphere Defense

We validated the performance of the sphere defense against Algorithm 1, Sasaki’s algorithm and our proposed algorithms. We calculated the ratio of poisoning data removed by the sphere defense for 36 poison generated by each algorithm. The result is in Table 2. As the results show, the sphere defense succeeded to remove most of poisoning data for ID1, 3, 4 in Algorithm 1. In contrast, not a single poison was removed in ID2. This implies that targeted malware of ID2 are distributed near the goodware, and hard to distinguish them. However, poisoning data can be easily detected by using the sphere defense for defense-unaware algorithms in many cases. We confirmed that results of removal ratio for Sasaki’s algorithm are almost 0% and defense-aware algorithms (Algorithms 3 and 4) are always 0% since these algorithms generate poison so that they are inside the sphere $\mathcal{F}_{\text{good}}$. For the effect on the attack success rate of the sphere defense, we will describe in the following subsection.

4.4 Attack Performance Against Sphere Defense

In the experiment, we compared the attack performance of previous algorithms (Algorithm 1, Sasaki’s algorithm and label-flip algorithm) and proposed algorithms (Algorithm 3 and Algorithm 4) for discrete and continuous dataset against the sphere defense. The results except for Algorithm 1 are shown in Fig. 1. `mal_B` stands for the targeted malware, `mal_NB` means non-targeted malware, and `good` is goodware in Fig. 1. The attack success rate of Algorithm 1 did not increase for ID1, 3 and 4 as the poisoning rate increased since almost

poisoning points were removed. The attack success rate reached at 39% with the poisoning rate 15% for ID2.

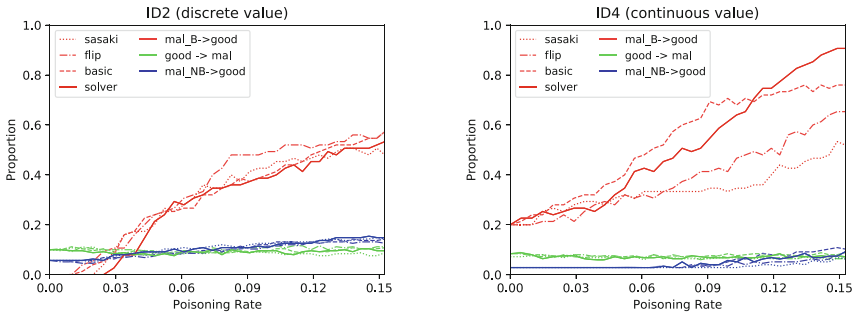


Fig. 1. The attack performance of Sasaki’s algorithm (**sasaki**), label flip algorithm (**flip**), Algorithm 3 (**basic**) and Algorithm 4 (**solver**) for discrete and continuous dataset against the sphere defense. The red lines correspond to the attack success rate ($\text{mal}_B \rightarrow \text{good}$), green lines correspond to the false positive rate of the goodwill ($\text{good} \rightarrow \text{mal}$), and blue lines correspond to the false negative rate of the nontargeted malware ($\text{mal}_{NB} \rightarrow \text{good}$) for each algorithm. (Color figure online)

For a discrete dataset (ID2), the attack success rate for each algorithm is similar. The label-flip algorithm seems to be a little higher than other algorithms for $\varepsilon < 0.03$ and $0.07 < \varepsilon$. The reason is that the feature vector \mathbf{v} of ID2 has binary value and optimization-based algorithms failed to move the vector \mathbf{v} to the desired coordinate since the perturbation $\mathbf{v} + \mathbf{d}$ is rounded to 0 or 1. The false negative rate of the nontargeted malware is about 0.05 for $\varepsilon = 0$ and 0.15 for $\varepsilon = 0.15$ for all algorithms. This implies that the decision boundary of the classifier is slightly displaced in the direction of nontargeted malware, depending on the amount of poison of targeted malware. The accuracy of the goodwill were almost unaffected by the poison. Similar results were obtained for ID1.

For a continuous dataset (ID4), Algorithm 3 has the highest attack success rate for $\varepsilon < 0.11$ and Algorithm 4 for $\varepsilon \geq 0.11$. Algorithm 4 reached 90.7% attack success rate for $\varepsilon = 0.15$. It is suggested that Algorithm 3 converges faster than Algorithm 4 with regard to the number of poisoning data. The label-flip algorithm has a lower attack success rate than Algorithm 3 and Algorithm 4 due to the fact that the poison generated by the label-flip algorithm is not optimized for targeted malware. Sasaki’s algorithm had the lowest attack success rate, which solves loss minimization problem and the smallest distance problem alternately. The false negative rate of the nontargeted malware is about 0.02 for $\varepsilon = 0$ and 0.09 for $\varepsilon = 0.15$ for all algorithms. The decision boundary between nontargeted malware and goodwill is affected by the number of poisoning data even in the continuous dataset. The accuracy of the goodwill was nearly constant. For ID3, we got the similar result.

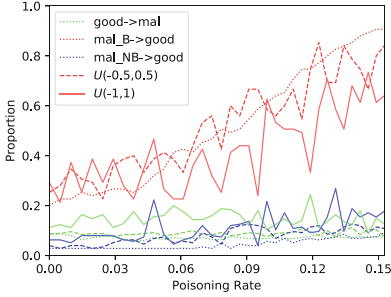


Fig. 2. The attack performance of Algorithm 4 with random noises drawn from $\mathcal{U}(-0.5, 0.5)$ and $\mathcal{U}(-1, 1)$. The dotted line is the original result ($\mathcal{U}(0, 0)$). The meaning of each color is the same as in Fig. 1. (Color figure online)

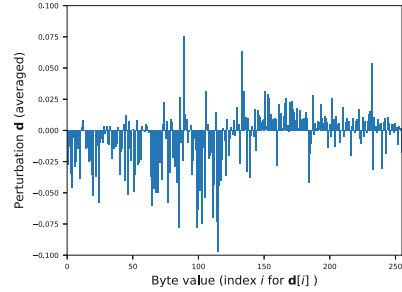


Fig. 3. Averaged value of the perturbation \mathbf{d} generated by Algorithm 4 for ID4 with respect to each byte value (dimension) for the perturbation radius $r = 0.5$.

4.5 Noise Resilience

We also evaluated whether our proposed method has noise resilience to preprocessing steps. Since Algorithm 4 adds a small perturbation to the feature vector, small changes to the perturbation could result in an ineffective attack. In practice, it often happens in preprocessing steps of the training phase. Also, a defender may intentionally add random noise to the training data to detect the poisoning data as experimented in [13]. To see the effectiveness of the random noise addition against Algorithm 4, we consider to add random noises \mathbf{n} drawn from the uniform distribution $\mathcal{U}(-a, a)$, $a \in \mathbb{R}$ to the training data $\mathbf{x} \in (\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{p}})$ before eliminating points using the sphere defense, *i.e.*, $\mathbf{x}' = \mathbf{x} + \mathbf{n}$ for $\mathbf{x} \in \mathcal{D}_{\text{train}}$ and $\mathbf{x}' = \mathbf{x} + \mathbf{d} + \mathbf{n}$ for $\mathbf{x} \in \mathcal{D}_{\text{p}}$. We set $a = 0.5$ and $a = 1$ since the perturbation radius r in Algorithm 4 is 0.5. The result is shown in Fig. 2. As for the attack success rate, the larger noise being added, the lower the accuracy for $\varepsilon > 0.05$. On the other hand, the accuracy increases for $\varepsilon \leq 0.05$. We infer that when ε is high, random noise \mathbf{n} negates the gradient of the perturbation \mathbf{d} to being goodware by the addition $\mathbf{d} + \mathbf{n}$. Whereas, when ε is low, noise rather works against (unpoisoned) targeted malware to counteract the gradient to be classified as malware. This phenomenon is also observed in the poisoning attacks on different dataset in [13]. For both the false positive rate of the goodware and the false negative rate of the nontargeted malware, the accuracy decreased as the noise range increased. Therefore, there is a trade-off between the whole accuracy of the model and the attack success rate with respect to the noise width.

Moreover, we observed the distribution of the perturbation \mathbf{d} to see the directionality of \mathbf{d} in each dimension. In the validation, we generate 100 poisoning data $\mathbf{x} + \mathbf{d}$ by the Algorithm 4 for ID4, then averaged \mathbf{d} with respect to each dimension. The perturbation radius $r = 0.5$ and the number of dimension $n = 200$. The theoretical average of the absolute values in each dimension is $r/\sqrt{n} \approx 0.035$. The

Table 3. Runtime and attack success rate for 36 poisons.

ID	Basic attack (Algorithm 3)		Label flip attack (Algorithm 4)		Attack using solver (Algorithm 4)	
	Success rate	Runtime [sec.]	Success rate	Runtime [sec.]	Success rate	Runtime [sec.]
1	12.0%	1,322.74	13.3%	1.27	16.7%	2.32
2	57.3%	1,504.66	57.3%	1.24	53.3%	2.27
3	73.3%	1,733.39	43.3%	1.08	66.6%	1.87
4	76.0%	1,087.94	65.3%	1.00	90.7%	1.90

result is shown in Fig. 3. Each dimension has obviously consistent directionality and different from a uniform distribution. If \mathbf{d} is distributed on a uniform distribution, then averaged value should be close to 0. Also, each dimension should have a positive gradient to being classified as goodware. Therefore, attack success rate is expected to change little for noises whose widths are nearly r/\sqrt{n} , and we confirmed it in the experiment. As is shown in Fig. 2, even if the noise width is considerably larger than $r/\sqrt{n} \approx 0.035$, poisoning attack still succeeds with some degree of attack success rate.

4.6 Runtime Comparison

Ultimately, we compared the computational costs of `basic` (Algorithm 3), `flip` and `solver` to generate poison data for each ID. In the experiment, we set $r = 0.5$ for `solver`, $burn = 2000$, $\eta = 0.1$ and $\varepsilon = 0.18$ ($|\mathcal{D}_p| = 36$). The results are shown in Table 3 with the attack success rate. From the result, Algorithm 4 is approximately 10^3 times faster than Algorithm 3. It is suggested that the cost required to solve the optimization problem of minimizing the loss by θ occupies the main part of the calculation time, and Algorithm 4 takes much less time since it needs to solve the problem only once regardless of the amount of poison data. The result also implies that embedding a solver in the label-flip algorithm can greatly improve the attack success rate without significantly increasing the calculation time. In realistic situations, poisoning attacks require a faster poisoning generator to perform the attack without noticing by the defender. For example, considering an attacker who is enrolled in a company that is outsourced the training of a machine learning model. Spending more time than necessary to generate the poison data may raise suspicion for the ordering company. The attack must be carried out quickly to keep them from noticing the presence of attacks.

5 Conclusion

In this paper, we focus on the problem of targeted poisoning attacks against malware detection. We apply the data sanitization defense against attacks, which is a commonly used technique for anomaly detectors to remove malicious data from the training set. We propose two targeted poisoning algorithms to generate poisons that evade sanitization: the basic variant and the improved variant. Our

streamlined algorithm is based on the combination of the label-flip attack [45] and the optimal attack strategy to maximize the training loss of the model, which was originally proposed for nontargeted attacks [42]. Experimental results confirmed that the sanitization is almost invalidated by our algorithms in attacker-favorable situations, while the defense is indeed effective against defense-unaware targeted poisoning attacks. Our algorithm achieved an the attack success rate of 91% by adding 15% poisoning, which is higher than the previous result of 85% for the same poisoning rate and the dataset without the data sanitization defense. Regarding the computational cost to generate poison data, our algorithm is approximately 10^3 times faster than the previous gradient-based algorithm.

References

1. Amos, B., Turner, H., White, J.: Applying machine learning classifiers to dynamic android malware detection at scale. In: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 1666–1671 (2013)
2. Anderson, H.S., Roth, P.: EMBER: an open dataset for training static PE malware machine learning models. arXiv preprint [arXiv:1804.04637](https://arxiv.org/abs/1804.04637) (2018)
3. Anindya, I.C., Kantarcioglu, M.: Adversarial anomaly detection using centroid-based clustering. In: 2018 IEEE International Conference on Information Reuse and Integration (IRI), pp. 1–8. IEEE (2018)
4. Baracaldo, N., Chen, B., Ludwig, H., Safavi, J.A.: Mitigating poisoning attacks on machine learning models: A data provenance based approach. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pp. 103–110 (2017)
5. Barreno, M., Nelson, B., Joseph, A.D., Tygar, J.D.: The security of machine learning. *Mach. Learn.* **81**(2), 121–148 (2010). <https://doi.org/10.1007/s10994-010-5188-5>
6. Biggio, B., Nelson, B., Laskov, P.: Poisoning attacks against support vector machines. arXiv preprint [arXiv:1206.6389](https://arxiv.org/abs/1206.6389) (2012)
7. Bleha, S., Slivinsky, C., Hussien, B.: Computer-access security systems using keystroke dynamics. *IEEE Trans. Pattern Anal. Mach. Intell.* **12**(12), 1217–1222 (1990)
8. Cen, L., Gates, C.S., Si, L., Li, N.: A probabilistic discriminative model for android malware detection with decompiled source code. *IEEE Trans. Dependable Secure Comput.* **12**(4), 400–412 (2014)
9. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint [arXiv:1712.05526](https://arxiv.org/abs/1712.05526) (2017)
10. Cretu, G.F., Stavrou, A., Locasto, M.E., Stolfo, S.J., Keromytis, A.D.: Casting out demons: Sanitizing training data for anomaly sensors. In: 2008 IEEE Symposium on Security and Privacy (Sp 2008), pp. 81–95. IEEE (2008)
11. Dai, J., Chen, C., Li, Y.: A backdoor attack against LSTM-based text classification systems. *IEEE Access* **7**, 138872–138878 (2019)
12. Diamond, S., Boyd, S.: CVXPY: a python-embedded modeling language for convex optimization. *J. Mach. Learn. Res.* **17**(83), 1–5 (2016)
13. Du, M., Jia, R., Song, D.: Robust anomaly detection and backdoor attack detection via differential privacy. arXiv preprint [arXiv:1911.07116](https://arxiv.org/abs/1911.07116) (2019)

14. Firdausi, I., lim, C., Erwin, A., Nugroho, A.S.: Analysis of machine learning techniques used in behavior-based malware detection. In: 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, pp. 201–203 (2010)
15. Gavriluț, D., Cimpoeșu, M., Anton, D., Ciortuz, L.: Malware detection using machine learning. In: 2009 International Multiconference on Computer Science and Information Technology, pp. 735–741 (2009)
16. Ham, H.S., Choi, M.J.: Analysis of android malware detection performance using machine learning classifiers. In: 2013 international conference on ICT Convergence (ICTC), pp. 490–495. IEEE (2013)
17. Hardy, W., Chen, L., Hou, S., Ye, Y., Li, X.: DL4MD: a deep learning framework for intelligent malware detection. In: Proceedings of the International Conference on Data Mining (DMIN), p. 61 (2016)
18. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system calls. *J. Comput. Secur.* **6**(3), 151–180 (1998)
19. Jung, W., Kim, S., Choi, S.: Poster: deep learning for zero-day flash malware detection. In: 36th IEEE Symposium on Security and Privacy, vol. 10, pp. 2809695–2817880 (2015)
20. Kang, P., Hwang, S., Cho, S.: Continual retraining of keystroke dynamics based authenticator. In: Lee, S.-W., Li, S.Z. (eds.) ICB 2007. LNCS, vol. 4642, pp. 1203–1211. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74549-5_125
21. Koh, P.W., Liang, P.: Understanding black-box predictions via influence functions. In: Proceedings of the 34th International Conference on Machine Learning, vol. 70. pp. 1885–1894. (2017) JMLR. org
22. Koh, P.W., Steinhardt, J., Liang, P.: Stronger data poisoning attacks break data sanitization defenses. arXiv preprint [arXiv:1811.00741](https://arxiv.org/abs/1811.00741) (2018)
23. Kolosnjaji, B., et al.: Adversarial malware binaries: Evading deep learning for malware detection in executables. In: 2018 26th European Signal Processing Conference (EUSIPCO), pp. 533–537. IEEE (2018)
24. Konečný, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. arXiv preprint [arXiv:1610.05492](https://arxiv.org/abs/1610.05492) (2016)
25. Kumar, B.J., Naveen, H., Kumar, B.P., Sharma, S.S., Villegas, J.: Logistic regression for polymorphic malware detection using anova f-test. In: 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), pp. 1–5. IEEE (2017)
26. Kwon, J., Lee, H.: Bingraph: Discovering mutant malware using hierarchical semantic signatures. In: 2012 7th International Conference on Malicious and Unwanted Software, pp. 104–111 (2012)
27. Laskov, P., Schäfer, C., Kotenko, I., Müller, K.R.: Intrusion detection in unlabeled data with quarter-sphere support vector machines. *PIK-praxis der Informationsverarbeitung und Kommunikation* **27**(4), 228–236 (2004)
28. Li, W.J., Wang, K., Stolfo, S.J., Herzog, B.: Fileprints: identifying file types by n-gram analysis. In: Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop, pp. 64–71. IEEE (2005)
29. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: defending against backdooring attacks on deep neural networks. In: Bailey, M., Holz, T., Stamatogiannakis, M., Ioannidis, S. (eds.) RAID 2018. LNCS, vol. 11050, pp. 273–294. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00470-5_13
30. Liu, Y., et al.: Trojaning attack on neural networks (2017)

31. McLaughlin, N. et al.: Deep android malware detection. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY 2017, p. 301-308. Association for Computing Machinery, New York (2017)
32. Muñoz-González, L., et al.: Towards poisoning of deep learning algorithms with back-gradient optimization. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pp. 27–38 (2017)
33. Newsome, J., Karp, B., Song, D.: Paragraph: thwarting signature learning by training maliciously. In: Zamboni, D., Kruegel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 81–105. Springer, Heidelberg (2006). https://doi.org/10.1007/11856214_5
34. Paudice, A., Muñoz-González, L., Lupu, E.C.: Label sanitization against label flipping poisoning attacks. In: Alzate, C. (ed.) ECML PKDD 2018. LNCS (LNAI), vol. 11329, pp. 5–15. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-13453-2_1
35. Rieck, K., Laskov, P.: Language models for detection of unknown attacks in network traffic. *J. Comput. Virol.* **2**(4), 243–256 (2007)
36. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M.: Microsoft malware classification challenge. arXiv preprint [arXiv:1802.10135](https://arxiv.org/abs/1802.10135) (2018)
37. Santos, I., Devesa, J., Brezo, F., Nieves, J., Bringas, P.G.: OPEM: a static-dynamic approach for machine-learning-based malware detection. In: Herrero, À. et al. (eds.) International Joint Conference CISIS' 12-ICEUTE' 12-SOCO' 12 Special Sessions. Advances in Intelligent Systems and Computing, vol. 189. Springer, Berlin (2013) https://doi.org/10.1007/978-3-642-33018-6_28
38. Sasaki, S., Hidano, S., Uchibayashi, T., Suganuma, T., Hiji, M., Kiyomoto, S.: On embedding backdoor in malware detectors using machine learning. In: 2019 17th International Conference on Privacy, Security and Trust (PST), pp. 1–5. IEEE (2019)
39. Sgandurra, D., Muñoz-González, L., Mohsen, R., Lupu, E.C.: Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. arXiv preprint [arXiv:1609.03020](https://arxiv.org/abs/1609.03020) (2016)
40. Shafahi, A., et al.: Poison frogs! targeted clean-label poisoning attacks on neural networks. In: Advances in Neural Information Processing Systems, pp. 6103–6113 (2018)
41. Siddiqui, M., Wang, M.C., Lee, J.: Data mining methods for malware detection using instruction sequences. In: Artificial Intelligence and Applications, pp. 358–363 (2008)
42. Steinhardt, J., Koh, P.W.W., Liang, P.S.: Certified defenses for data poisoning attacks. In: Advances in Neural Information Processing Systems, pp. 3517–3529 (2017)
43. Tran, B., Li, J., Madry, A.: Spectral signatures in backdoor attacks. In: Advances in Neural Information Processing Systems, pp. 8000–8010 (2018)
44. Wang, B., et al.: Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 707–723. IEEE (2019)
45. Xiao, H., Xiao, H., Eckert, C.: Adversarial label flips attack on support vector machines. In: ECAI, pp. 870–875 (2012)
46. Yuan, Z., Lu, Y., Wang, Z., Xue, Y.: Droid-sec: deep learning in android malware detection. In: Proceedings of the 2014 ACM conference on SIGCOMM, pp. 371–372 (2014)



STDNeut: Neutralizing Sensor, Telephony System and Device State Information on Emulated Android Environments

Saurabh Kumar^(✉), Debadatta Mishra, Biswabandan Panda,
and Sandeep K. Shukla

Indian Institute of Technology, Kanpur, India
{skmtr,deba,biswap,sandeeps}@cse.iitk.ac.in

Abstract. Sophisticated malware employs various emulation-detection techniques to bypass the dynamic analysis systems that are running on top of virtualized environments. Hence, a defense mechanism needs to be incorporated in emulation based analysis platforms to mitigate the emulation-detection strategies opted by malware. In this paper, first we design an emulation-detection library that has configurable capabilities ranging from basic to advanced detection techniques like distributed detection and GPS information. We use this library to arm several existing malware with different levels of emulation-detection capabilities and study the efficacy of anti-emulation-detection measures of well known emulator driven dynamic analysis frameworks. Furthermore, we propose STDNeut (**S**ensor, **T**elephony system, and **D**evice state information **N**eutralizer) – a configurable anti-emulation-detection mechanism that defends against the basic as well as advanced emulation-detection techniques regardless of which layer of Android OS the attack is performed on. Finally, we perform various experiments to show the effectiveness of STDNeut. Experimental results show that STDNeut can effectively execute a malware without being detected as an emulated platform.

Keywords: Android · Malware · Security · Sandbox · Emulation-detection

1 Introduction

Mobile platforms like Android are common in modern-day devices because of its open-source availability and robust support for mobile application (App) development. According to a recent report published by International Data Corporation (IDC), the global market share of Android OS was 85.1% [16] in the year 2018. As a consequence of such a large scale adoption of Android and ever-increasing contributions in the Android App space, the security of these devices has become a non-trivial challenge recently. A study related to malware activities in the Android platform published by G DATA shows that in the first half of 2018, more than 2 million new Android malware were recorded. In other words, an Android malware is born in every seventh second [3].

Existing approaches that address the security issues arising due to the rapid growth of Android malware can be broadly classified into two categories: static analysis based techniques and dynamic analysis/detection based techniques [30]. Techniques that are based only on static analysis [11, 20, 38] are insufficient to address the security issues presented by the malware especially designed to bypass the static analysis based defenses. For example, advanced malware employ techniques such as dynamic code loading, native code exploitation, Java-reflection mechanisms, and code encryption to bypass static analysis based detection [33]. In order to address the limitations of static analysis techniques, dynamic analysis techniques are preferred. While dynamic analysis is widely used, the existing frameworks fall short in tackling platform sensing malware which is a reality today as emulator-based analysis platforms are used as opposed to real devices for cost-effectiveness.

The Problem: Many malware [36] employ techniques to detect the underlying emulation platform before showing their true behavior. To the best of our knowledge, none of the existing emulator driven dynamic analysis frameworks make claims regarding their effectiveness towards nullifying possible emulation-detection adopted by malware.

One of the root causes of the problems related to emulation-detection is heavy usage of emulated platforms by dynamic analysis solutions. Many dynamic analysis systems (Droidbox [21], MobSF [24], CuckooDroid [34], DroidScope [40], CopperDroid [32], and Bouncer [22]) are based on virtualized environments to perform malware analysis by executing the Apps in a controlled environment and collect various event logs for further analysis. As a negative consequence, malware developers utilize various emulation-detection techniques to detect the underlying execution environment and adapt their behavior accordingly.

Identifying the underlying execution environment by a malware is shown to be possible by many previous studies [18, 25, 27, 36]. Recently, it has been shown that the dynamic analysis performed for identifying malicious Apps by the Google Bouncer (a dynamic analysis system deployed on Play Store) [22] can be bypassed by detecting the underlying execution environment [25, 27]. Vidas et al. [36] present generic emulation-detection approaches that can be used to evade dynamic analysis, whereas Morpheus et al. [18] show more than 10000 heuristics to detect underlying emulated platforms. In contrast, DroidBench-3.0 [8], an open test suite for evaluating the effectiveness of an analysis system includes a subset of small Apps (based on [36]), which can help in analyzing the effectiveness of dynamic analysis frameworks. However, the emulation-detection mechanisms used by DroidBench-3.0 Apps are very basic and many dynamic analysis frameworks (e.g., CuckooDroid [34], Droidbox [21]) have already incorporated anti-emulation-detection measures in their design. Even though the dynamic analysis frameworks are capable of providing defense mechanisms against rudimentary emulation-detection, *malware developers find new ways to detect the underlying execution environment at runtime* [35]. For example, Google Play Protect which is used to certify Android Apps, fails to detect malware that spread across 85 different Apps affecting nine million Android devices [26].

Our Goal: We believe, a dynamic analysis system should provide a configurable anti-emulation-detection mechanism, so that a smart malware developer would find it difficult to evade the dynamic analysis by studying the analysis framework. Moreover, we would like to emphasize the need for a validation mechanism to understand the effectiveness of the same.

Our Approach: As a validation mechanism, we design a pluggable emulation-detection library with configurable levels of emulation-detection capabilities, which can be incorporated by any malware. We use this library to arm several existing malware with different levels of emulation-detection capabilities and study the efficacy of anti-emulation-detection measures of well known dynamic analysis frameworks. Further, using the findings of our analysis, we develop STDNeut (**S**ensor, **T**elephony system, and **D**evice state information **N**eutralizer), a detailed anti-emulation-detection system fully designed using Qemu [9] based Android emulator [5].

A robust and extensible validation framework can provide the basis for understanding the effectiveness of existing dynamic analysis systems w.r.t. their anti-emulation-detection measures. Moreover, the framework should provide guidance principles for designing new dynamic analysis systems with detailed anti-emulation-detection measures. Towards these objectives, our contributions are as follows:

(i) We design an emulation-detection library encompassing several advanced detection techniques like distributed detection and GPS information (Section 3.1). We use this library to perform an empirical evaluation of existing dynamic analysis frameworks against the basic and extended emulation-detection techniques (Sect. 3.2). The library can be configured with varying levels of emulation-detection methods and can be embedded into different malware in a seamless manner.

(ii) We propose STDNeut by using the insights of the empirical validation of existing frameworks (Sect. 4) that remain undetected even if the emulation-detection is performed at any layer of the Android OS w.r.t. sensors, telephony system and device state. Further, we show the effectiveness of STDNeut in neutralizing different emulation-detection techniques (Sect. 5). Note that, detection of an analysis framework by observing a rooted device or the existence of an instrumentation framework like Xposed is beyond the scope of this paper.

2 Background and Related Work

In this section, we discuss the Base Transceiver Station (BTS) as a smartphone frequently communicates with it. After that, we provide a brief overview of the emulation-detection followed by the related work.

2.1 Base Transceiver Station

BTS [23] is a piece of wireless communication equipment that establishes communication between a mobile device and a network. The BTS is associated with

a base station ID that uniquely identifies a BTS worldwide. Base station ID comprises of four components: *(i)* mobile country code (MCC), *(ii)* mobile network code (MNC), *(iii)* location area code (LAC), and *(iv)* a cell ID (CID). A combination of these gives a unique identity to a BTS. Several commercial and public services are available which provide the geo-location of a cell by submitting its station’s unique ID.

2.2 Emulation-Detection

The primary issue with an emulated system is its inability to replicate a complete system that matches the exact configuration and characteristics of a physical device. The core idea of emulation-detection is to observe the differences between virtual and physical machines using a program to identify the underlying infrastructure. Vidas et al. [36] and Morpheus [18] have shown that such differences can be used to detect underlying emulated platforms through a stand-alone App. Vidas et al. [36] propose a few generic detection methods based on the device characteristics, e.g., differences in hardware components (like sensors and CPU information) and software components (like Google’s Apps are not present). Morpheus [18] presents more than 10000 heuristics to detect the underlying emulated platform which has broadly classified it into three categories viz. i) Files, ii) APIs, and iii) System Properties related detection which are similar to the techniques proposed in [36]. The emulation-detection methods shown in [18,36] fall in the category of basic emulation-detection, and most of the dynamic analysis systems are capable of bypassing them.

2.3 Related Work

Static analysis techniques fail to capture the precise characteristics of an App because of the advanced App development techniques like dynamic code loading and reflection [33]. This has led to the introduction of a dynamic analysis of Android Apps. Dynamic analysis techniques execute an App in a controlled environment called “sandbox” which can be a real device or an emulated platform to observe its behavior. Dynamic analysis on a real device is costly and incurs significant overhead [31]; hence, an emulator driven sandbox gets the attention of security researchers.

Emulation driven analysis tools must provide the ability to hide the emulated environment from the target App along with the profiling features. In the absence of such defense mechanisms, an App can evade the dynamic analysis by detecting the emulated platform [18,25,27,36].

The techniques in [25,27] use API based detection and IP address based detection to evade dynamic analysis on Google Bouncer, which essentially works by determining whether the IP address belongs to Google or not.

DroidBench [8], a recent work, provides a set of Apps to detect the underlying virtual environment based on the methods proposed in [18,36]. Further, DroidBench also introduces some new methods that utilize the call history and number of contacts in emulation-detection. Similarly, Caleb Fento [13] and

Gingo [14] have developed stand-alone Apps to detect Android virtual devices. Caleb Fento [13] uses the information shown by Vidas et al. [36] as a detection method to detect Google’s Android emulator. On the other hand, Gingo [14] extends the same to detect custom virtual devices (like Genemotion, Nox player) along with Google’s Android emulator.

Other than the stand-alone Apps discussed so far, some android libraries [7, 17, 19] have also been developed to detect emulated Android devices which can be integrated with any App. Libraries [7, 17] use similar information as presented in [14], whereas the library [19] uses the accelerometer data in the detection mechanism.

Additionally, Diao et al. [12] proposed an approach to evade runtime analysis by differentiating a user from a bot by analyzing the interaction pattern. This detection technique is inclined to differentiate user from a bot to bypass runtime analysis and does not focus on the emulation-detection.

Costamagna et al. [10] have shown the evasion of Android sandbox through the fingerprinting of usage-profile. This technique works by observing the device usage information like SMS, call history and others which remain the same when multiple samples of a malware family execute inside a sandbox. However, the information received at the server from numerous Apps (malware sample of the same family) during different executions remains identical. The same information is fed to the next subsequent malware sample to evade the dynamic analysis.

Existing sandboxes [21, 24, 34, 40] provide some anti-emulation-detection measures to mitigate the emulation-detection attack. For example, Droid-Box [21] modifies the Android Open Source Project (AOSP) to bypass the emulation-detection, while some others [24, 34] utilize the hooking framework (like Xposed [39]) and provide static but realistic information. Though, they can defend against the basic emulation-detection in DroidBench but they do not work in the context of extended emulation-detection.

Some other anti-emulation-detection works have also been proposed that modify the targeted App before submitting it for analysis [29, 37]. Siegfried et al. [29] use the backward slicing method and remove the emulation-detection related checks from an App. On the other hand Droid-AntiRM [37] performs bytecode instrumentation to defeat the emulation-detection. In both approaches, an App needs modification before submission for dynamic analysis. Thus, the integrity of an App is lost through such changes.

3 Motivation

To study the effectiveness of the existing dynamic analysis frameworks, we require a tool with varying levels of the emulation-detection method. In this section, first, we give an overview of the flexible emulation-detection library that we have designed with a collection of emulation-detection methods beyond the basic detection techniques (see Sect. 2.2). We use this library to evaluate the existing frameworks about their anti-emulation-detection measures empirically. At last, we present the insights learned from this evaluation in designing STDNeut.

Table 1. Classification of emulation-detection techniques.

Detection categories	Description
Unique device Information (basic)	Detection by observing unrealistic device information values (e.g., IMEI value is 00000)
Unique device Information (smart)	Detection based on fixed reading of unique device information (e.g., IMEI value is constant)
Sensors reading	Absence of sensor or observing static values from fluctuating sensors (e.g., fixed reading of Light sensor)
Device state information	No change to the device state w.r.t. telephony signal, battery power
GPS information	No change on GPS location data or fake location change
Distributed detection	Observing identical unique information for multiple devices in a network

3.1 Overview of Emulation-Detection Library (EmuDetLib)

As a validation mechanism, we have developed a flexible **emulation-detection library** (EmuDetLib). The detection techniques in EmuDetLib can be broadly classified into five categories (refer Table 1): *(i)* Unique device information (UDI), *(ii)* Sensors reading, *(iii)* Device state information, *(iv)* GPS information, and *(v)* Distributed detection.

Unique Device Information: This method uses information like IMEI (international mobile equipment identity) and IMSI (international mobile subscriber identity), that is unique to a device and employs basic and smart methods to detect an emulated environment. In basic detection, EmuDetLib observes any unrealistic data (not in the prescribed format) obtained from the device, whereas in smart detection, the library also checks whether the information is static or not by comparing it against known static values of different frameworks.

Sensors Reading: Nowadays smartphones have various sensors for different purposes that can be broadly classified into two categories—motion sensors and environmental sensors. As the data observed on these sensors fluctuate continuously, this insight can be used to detect the underlying emulated environment. A recent example of sensor-based detection is the observation of TrendMicro, where malware (in Play Store) make use of the motion-detection feature to evade the dynamic analysis [35]. This method detects an emulated environment by utilizing sensors count and/or by observing fixed sensor values from fluctuating sensors.

Device State Information: In reality, a device state gets changed due to some internal/external event such as change in telephony signal strength, battery power and incoming SMS/Calls. However, such state changing behavior is missing in an emulated environment. Our library observes these information to detect the underlying emulated environment.

Using the GPS Location Information: GPS is also a sensor and malware can use similar methods (as explained above) that are used for other sensors to detect emulated platforms. However, the emulation-detection based on the GPS is somewhat different from other sensors, as explained below.

Android provides rich APIs to perform various tasks. One such API gives the power to generate a mock location that can be used by an App to introduce a fake location when queried. An Android App requires `ACCESS_MOCK_LOCATION` permission to use the mock location API. The other source for geo-location is BTS ID. Android provides API to query BTS ID, and we can get its geo-location by using publicly/commercially available services (<https://opencellid.org>). Hence, the geo-location-based emulation-detection technique only works when one of the following conditions is satisfied: (i) there is no change in the geo-location of the device, (ii) the mock location API is used to set the geo-location of the device, or (iii) BTS geo-location is not collaborating with the GPS location.

Distributed Emulation-Detection: Nowadays, most Apps require communication with a centralized server to share their status or get new information. To identify a device uniquely at the server, an App typically generates a unique ID called an AppID. A smartphone also contains device-related unique IDs namely IMEI, IMSI, SIM Serial number, and others. These information can also help in identifying a device uniquely at the server as explained below.

It is trivial to see that a slightly different malware in terms of its signature can be generated easily by changing its package name, altering the function name and variable naming convention, or by introducing dummy code while retaining the overall functionality and the server address. Such malware can communicate the unique device information to a remote server to identify the emulated environment remotely. In this situation, the emulation-detection can happen at the server by querying the device information from the connected devices. If a server detects that multiple devices have identical information (expected to be unique), it can flag those devices as emulated environment. As this emulation-detection is carried out in the context of multiple connected devices, we classify this detection technique as a distributed emulation-detection.

The emulation-detection methods in EmuDetLib discussed above are configurable and any App can change its detection mechanism by creating a suitable configuration file. For more details on EmuDetLib and ethical concern, refer weblink: <https://skmtr1.github.io/EmuDetLib.html>.

3.2 Evaluation of Existing Frameworks

To perform empirical evaluation of the existing dynamic analysis frameworks, we have integrated EmuDetLib into the DroidBench-3.0 [8] benchmark Apps (referred to as EmuDetLib-Bench). Apart from the EmuDetLib-Bench, we have collected 1000 malware where the dex date is of the year 2019 from AndroZoo [4] along with the motion sensor’s malware disclosed by the Trend Micro [35] (referred to as RealMal) to evaluate the existing frameworks. We have considered

Table 2. Evaluation of existing framework against detection library EmuDetLib.

Detection type	Sub-type	Emulator	Droidbox	CuckooDroid	MobSF
Unique Device Information	Basic	✓	×	×	×
	Smart	✓	✓	✓	✓
Sensors	Count	×	×	×	✓
	Reading	✓	✓	✓	✓
Device State	–	✓	✓	✓	✓
GPS	Cond (i) (Normal)	✓	✓	✓	✓
	Cond (i) (Fake)	×	×	×	×
	Cond (i) & (ii)	✓	✓	✓	✓
	Cond (i) & (iii)	✓	✓	✓	✓
Distributed (Server config)	No Emulation	×	×	×	×
	W/- Emulation	✓	✓	✓	✓

Note: ✓ represents successful detection of underlying emulation environment, whereas × represents failure in detecting emulation environment. We use this notation in the rest of the tables. In GPS based detection, “Fake” represents a sandbox executing fake GPS location generating App/service. Normal represents without fake location App/service, and rest of the condition is evaluated with both the setting i.e. fake and without fake app. In distributed emulation, no Emulation represent the server without emulation-detection algorithm whereas W/- Emulation represent server deployed with emulation-detection algorithm.

CuckooDroid [34], Droidbox [21], and MobSF [24] along with the vanilla Android emulator (referred to as emulator) [5] as the candidate analysis systems for the empirical study, as they are readily available. We exclude the online analysis systems and other sandboxes in this study. The main reason is that an online analysis system has a long waiting queue and takes a longer time to schedule a sample for the evaluation. Hence, these frameworks are not preferred for this evaluation.

Further, to evaluate GPS information based detection and distributed detection, we need a different environment. For GPS, we require a fake GPS location generation app inside an emulated device. For distributed detection, we need a server where the emulation-detection method is deployed and requires multiple instances of the same sandbox running at the same time. We utilize the command and control server of the real malware Dendroid [28] by employing the emulation-detection algorithm (see Algorithm 2 at weblink: <https://skmtr1.github.io/EmuDetLib.html#a2>).

Table 2 shows the evaluation result of the emulation-detection of candidate sandbox against all the detection methods shown in Table 1. In Table 2, the sub-type represents the subcategory/configuration of the evaluation. As shown in Table 2, in distributed detection, when the server is configured with the emulation-detection method, none of the frameworks can hide their emulated environment. Similarly, in GPS-based detection, only with a fake app installed emulated-platform can bypass the detection mechanism in condition 1 (refer

to GPS information based detection). In other cases of GPS, the sandbox is flagged as an emulated platform by the detection library. There is one other case in the sensors category with count where MobSF is the only sandbox that cannot bypass the detection mechanism. The reason being, it is designed on top of VirtualBox and does not support sensors. In contrast, all other sandboxes use an Android emulator, which comprises of 7–8 sensors inbuilt and bypasses the emulation-detection based on sensor count.

Similarly, on executing samples of RealMal (see Table 2 at weblink: <https://skmtr1.github.io/EmuDetLib.html#t12> for classification and evaluation), Android SDK emulator cannot hide its emulated environment against malware samples with emulation-detection capability. Simultaneously, other sandboxes get detected by the malware samples under the category of device state and sensors. To reason about such behavior, we have investigated BatterySaverMobi malware from RealMal samples (see Listing 1 at weblink: <https://skmtr1.github.io/EmuDetLib.html#cs1> for code snippet), which uses accelerometer (line 5) reading to observe motion on a device. If any motion takes place, then it executes the malicious code (line 15). Hence, Such malware can bypass the dynamic analysis job performed on existing sandboxes.

3.3 Summary of Emulation-Detection

Some key observations regarding the effectiveness of anti-emulation-detection measures of the existing analysis platforms against EmuDetLib are shown below.

- i)* Existing analysis frameworks are able to bypass the basic emulation-detection techniques based on unique device information. However, they fail to defend when the emulation detection attacks are performed by analyzing the underlying defense mechanism. The main reason being either the data is unrealistic (basic detection) or the data is realistic but static (smart attack).
- ii)* Each framework fails to defend against the emulation-detection attacks based on fluctuating sensors and GPS data since the data does not represent the realistic behavior of a device.
- iii)* Similar to the detection methods based on UDI, existing frameworks are also not able to defend against distributed emulation-detection. The observation of similar data for unique device-related information across multiple devices helps in raising the red-flag regarding the underlying emulated environment.
- iv)* Detection methods based on the device state (e.g. Telephony, Battery Power) also successfully detect the underlying emulated environment due to the absence of defense mechanisms in the analysis frameworks.

In short, the extended emulation-detection techniques show that the existing publicly accessible dynamic analysis frameworks do not provide foolproof anti-emulation-detection measures. Therefore, there is a need for a robust anti-emulation-detection approach that can hide the underlying platform from smart emulation-detection measures. Note that the emulation-detection techniques can

also utilize the timing channel to detect the emulated platform (like timing measures against the graphics subsystem). Such heuristics require a sufficient number of events to understand the underlying execution environment, which tends to increase their code footprints and flag such an App as abnormal. Due to this limitation, we do not discuss any timing channel based emulation-detection methods. In the next section, we discuss the design and implementation of STDNeut which incorporates a robust anti-emulation-detection system.

4 STDNeut: Design and Implementation

In this section, first we discuss the process of generating realistic sensor’s data and the challenges associated with it. After that we provide an overview of STDNeut, a detailed anti-emulation-detection system and elaborate on the design of its various components. STDNeut aims to neutralize emulation-detection using different sensors, telephony system, and device state data.

4.1 Realistic Sensor Data Generation

A smartphone contains multiple sensors (e.g., accelerometer, GPS, and others) or interacts with an external entity like BTS. A malware can use these sensors to detect an emulated environment. To nullify the effect of sensors based emulation-detection, we have identified three main challenges as follows:

- (i) Existing sensors value should fluctuate with respect to time.
- (ii) Detection of emulation environment through sensor correlation.
- (iii) Model should be flexible to incorporate new sensors and sensor relations.

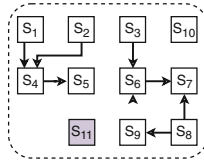


Fig. 1. An example of sensor’s dependency graph. Sensor S_{11} in shaded box represents a new sensor introduced in the system.

To better understand these challenges, let us take a directed graph shown in Fig. 1 that represents eleven sensors (S_1 to S_{11}), and influence of one sensor on others in terms of driving the sensor’s values. An arrow from sensor S_i to sensor S_j denotes that the value of sensor S_j depends on the value of sensor S_i . If we see an update in the value of sensor S_i , then sensor S_j ’s value should also be seeking an update according to S_i ’s value. As shown in Fig. 1, some sensors do not depend on other sensors (sensor with zero in degree); we name them as independent sensors, whereas sensors with in degree ≥ 1 are called dependent sensors because the value of these sensors depends on the value of others.

Challenge (i) is easy to understand, which states that the value of the sensor should fluctuate concerning time. For example, let us consider sensor S_{10} (assuming as a light sensor) in Fig. 1, the value of this sensor should be updated according to the operating environment lighting condition. Similarly, other sensor's value should also be updated w.r.t. time or working environment condition.

To understand challenge (ii), consider two sensors S_4 (assuming as GPS) and S_5 (assuming as BTS). As shown in Fig. 1, sensor S_5 's value depends on the value of sensor S_4 . This dependency is based on the distance between the values of S_4 and S_5 , which cannot be more than x meters. This x may vary depending on the area density (population and obstacles) of the BTS. Further, to be more clear about challenge (ii), let us include two more sensors S_1 (as time) and S_2 (as an accelerometer). The value of sensor S_4 depends on both the sensors, i.e., S_1 and S_2 . If we consider time and GPS, then there is a correlation between the current GPS location and the previous location w.r.t. time elapsed. For example, if the current GPS location is Washington DC, a person cannot reach New York in five minutes. Similarly, when considering accelerometer and GPS, then the measurement of the distance travelled through accelerometer should match with the distance between two consecutive GPS locations. Hence, a sensor-based anti-emulation-detection system should be compliance to all these scenarios so that the use of sensor's value in an innovative way (as described above) cannot reveal the identity of the underlying system.

Challenge (iii) is related to the introduction of a new sensor into the system. If a new sensor is included in the system, either it is an independent or dependent sensor (sensor S_{11} as shown in Fig. 1), the system should be flexible to reprogram so that new sensors can also be adopted for providing anti-emulation-detection capability.

To emulate realistic values for sensors, one should consider all the scenarios, as discussed above. Hence, a fine-grained method is needed to emulate sensors reading while maintaining the dependencies between them along with the re-programmable capability to adopt new sensors in the system.

To address all the challenges as mentioned above, we present Algorithm 1, which takes two lists. One list holds the available sensor object ($sensors_{obj}$) and the other is related to the dependency between sensors ($dependSens_{obj}$). A sensor's object comprises of sensor's identity (like accelerometer, GPS), a default handle and the initial value. The default handle is useful when a sensor does not depend on others (independent sensors), and the initial value is used to initialize the sensor. On the other hand, a dependency object comprises the identity of two sensors S_i and S_j , and a dependency function F_{ij} , which represents the dependency between S_i and S_j . These two lists have to be provided by a user, and Algorithm 1 generates an ordered list of sensors handle ($sensors_{hdl}$), which can be executed at the analysis time to emulate the sensor's value while preserving the relationship between them.

Algorithm 1: Generate Handle for Sensors

```

Input :  $sensors_{obj}, dependSens_{obj}$  // List of sensors and dependencies objects
Output:  $sensors_{hndl}$  // Ordered list of handles to generate realistic sensors values

1  $sensors_{hndl} \leftarrow \phi$ 
2  $Unprocessed_{chld} \leftarrow \phi$  // Sensors queue whose child is not processed
3  $Processed_{chld} \leftarrow \phi$  // List of sensors whose child is already processed
4  $Dependency_{graph} \leftarrow generate\_graph(dependency_{obj}, sensors_{obj})$ 
5  $Independent_{sensors} \leftarrow getZeroInDegreeNodes(Dependency_{graph})$ 
6 foreach  $S$  in  $Independent_{sensors}$  do
7    $S_{hndl} \leftarrow default_{hndl}(sensors_{obj}, S)$ 
8    $append(sensors_{hndl}, (S, S_{hndl}))$ 
9    $append(Unprocessed_{chld}, S)$ 
10 while  $\neg(empty(Unprocessed_{chld}))$  do
11    $S \leftarrow dequeue(Unprocessed_{chld})$ 
12    $chlds \leftarrow getChilds(Dependency_{graph}, S)$ 
13   foreach  $C$  in  $chlds$  do
14      $dep_{func} \leftarrow getDep_{func}(dependency_{obj}, (S, C))$ 
15      $C_{hndl} \leftarrow generate_{hndl}(sensors_{obj}, C, dep_{func})$ 
16     if  $C$  not in  $sensors_{hndl}$  then
17        $append(sensors_{hndl}, (C, C_{hndl}))$ 
18     else if  $C$  is in  $Processed_{chld}$  then // Handling cyclic dependency
19        $dep_{func} \leftarrow getDep_{func}(dependency_{obj}, (S, C))$ 
20        $C_{hndl} \leftarrow generate_{hndl}(sensors_{obj}, C, dep_{func})$ 
21        $update_{hndl}(sensors_{hndl}, (C, C_{hndl}))$ 
22     else
23        $update_{hndl}(sensors_{hndl}, (C, C_{hndl}))$ 
24     if  $C$  not in  $Unprocessed_{chld}$  and  $C$  not in  $Processed_{chld}$  then
25        $append(Unprocessed_{chld}, C)$ 
26    $append(Processed_{chld}, S)$ 
27 return  $sensors_{hndl}$ 

```

In Algorithm 1, $Unprocessed_{chld}$ denotes a queue of sensors whose immediate child needs processing w.r.t. its handle to emulating the sensor value, whereas $Processed_{chld}$ holds the list of sensors whose child has already been processed. Apart from storing processed sensors, the algorithm utilizes this list to break any cyclic dependency (see dependency among sensors S_6 to S_9 in Fig. 1), which is a rare case for sensors. As shown in line 4, the algorithm generates a dependency graph among sensors by using the list of $dependSens_{obj}$ and $sensors_{obj}$. Line 5 gets the list of independent sensors from the dependency graph from where actual learning of sensor handle starts. From lines 6 to 9, the algorithm obtains a handle for each independent sensor, which is equivalent to the default handle in sensor object. The default handle is used to generate the value for a sensor, which does not depend on other sensors. Apart from the sensor handle, independent sensors are then appended in the $Unprocessed_{chld}$ queue, because the children of these sensors may require a handle.

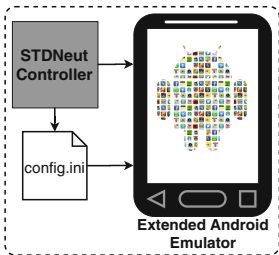
From lines 10 to 26, the algorithm generates the handles for the dependent sensors. The algorithm terminates when the $Unprocessed_{chld}$ queue does not contain any sensor for processing. Line 14 gets the dependency function between parent sensor S and the child sensor C by using the $dependSens_{obj}$ and a handle gets generated at line 15. At line 16, it checks if the sensor is not in the list of $sensors_{hndl}$, algorithm directly adds this handle into $sensors_{hndl}$. In other cases, it updates the already learned handle based on the current dependency and the

dependency learned earlier. For updating an already learned handle, there can be two possibilities, one is related to cycle (see cyclic dependency in Fig. 1 among sensors S_6 to S_9) and the other is when a sensor depends on more than one sensor (See sensor S_4 in Fig. 1). A cyclic dependency is resolved at line 18 in Algorithm 1, where a new dependency function is calculated between parent S and child C . To obtain the new dependency function, we utilize the last value of S (referred to as \bar{S} in line 19) to update the handle of C . At last, when all the children of a sensor S are processed, S is added to the $Processed_{chld}$ at line 26. Finally, the algorithm returns an ordered list of $sensors_{hdl}$, which is then used to emulate the sensor's value at run-time. This algorithm handles the challenge (i) and (ii). For challenge (iii), if the user updates the list of sensor objects and dependency objects, then it re-generates the sensor handles for all the sensors, including the new sensors.

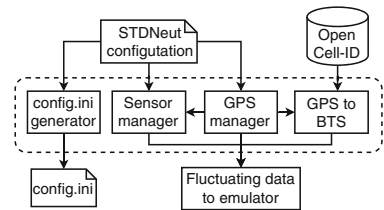
4.2 STDNeut Overview

STDNeut system provides robust support for anti-emulation-detection that can be used to design an efficient framework for malware analysis. Fig. 2 shows the architecture of STDNeut along with the design of its controller. As shown in Fig. 2(a), there are two main subsystems of the STDNeut: (i) Extended Android Emulator and (ii) STDNeut Controller (see Fig. 2(b)).

Extended Android Emulator: It is responsible for spoofing the information related to sensors, telephony systems, and device data. The STDNeut controller and `config.ini` file govern this spoofing information to the Android emulator. Most of the device-specific information, like IMEI, remains constant during the execution time, while the values for sensors and telephony signal fluctuate over time. During the boot time, the Android emulator reads `config.ini` file and configures a virtual device with device-specific information that is unique to it, while the STDNeut controller handles the fluctuating values at run-time.



(a) STDNeut design overview.



(b) Design of STDNeut controller.

Fig. 2. Architecture of STDNeut, an anti-emulation detection system along with the STDNeut controller.eps

STDNeut Controller: It is responsible for launching an App inside the emulator and feeding essential information for anti-emulation-detection. For example, the controller generates a `config.ini` file that is being used by the Android emulator to configure a virtual device with unique values. The controller also manages the hardware/environment generated events that alter the state of an Android device such as available sensors, telephony signal, and many more. This is achieved by frequently feeding-in realistic sensor data while maintaining the correlation with other sensors (as described in Sect. 4.1 by utilizing Algorithm 1) and other hardware related events into the emulator. To feed the sensor data and hardware-related events, the controller uses the emulator console APIs [6]. Other than the core features mentioned above, the controller also enables and configures other functionalities which simulate incoming calls/SMSes, manipulates signal strength, and many more. We discuss the extension made to Android emulator in the next section.

4.3 Extensions to the Android Emulator

A smartphone contains multiple sources of information that are either unique to a device and does not change during its life or information may get changed over time due to the operating environment that alters its state. Mostly, a device gets a unique identity from the telephony system that includes IMEI, IMSI, phone number, and many more. To interact with the telephony system, we use AT commands [1]. To provide a unique identity to a virtual device, we intercept the AT command request at the emulator layer for spoofing the response. For example, a smartphone makes “AT+CGSN” and “AT+CIMI” commands to query IMEI and IMSI numbers, respectively. This spoofed information is fed to the AT command by concerning the `config.ini` file. Similarly, in response to AT command, other values are also fed that remain constant but unique to a device. Apart from the `config.ini` file, these values can also be supplied to a virtual device using command line arguments. We use the emulator console to supply realistic data periodically for the hardware/environment events that alter the device state. The Android emulator provides most of the hardware like sensors, GPS, signal strength, and others; the data for them can be fed using emulator console. Android emulator does not provide any interface to change the BTS information with whom a device is currently associated. To provide a realistic GPS location, the information about the BTS associated with the device should collaborate. With this observation, we have added the BTS interface through the emulator console, and the STDNeut controller is supplying the realistic BTS identity.

Algorithm 2: Path patching for GPS trajectory

```

Input :  $Lat_{src}, Long_{src}, Lat_{dst}, Long_{dst}, nSteps$ 
Output: trajectory
1 trajectory  $\leftarrow \phi$ 
2  $LatStep_{max} \leftarrow |Lat_{src} - Lat_{dst}| / nSteps \times 2$ 
3  $LongStep_{max} \leftarrow |Long_{src} - Long_{dst}| / nSteps \times 2$ 
4  $Direct_{lat} \leftarrow +1$  if  $Lat_{dst} > Lat_{src}$  else  $-1$  // direction
5  $Direct_{long} \leftarrow +1$  if  $Long_{dst} > Long_{src}$  else  $-1$ 
6  $(lat, long) \leftarrow (Lat_{src}, Long_{src})$ 
7 append(trajectory, (lat, long))
8 foreach  $i$  in range(0, nSteps) do
9   |  $lat \leftarrow lat + rnd.uniform(0, LatStep_{max}) \times Direct_{lat}$ 
10  |  $long \leftarrow long + rnd.uniform(0, LongStep_{max}) \times Direct_{long}$ 
11  | append(trajectory, (lat, long))
12 return trajectory

```

4.4 STDNeut Controller

The primary responsibility of STDNeut controller is to generate `config.ini` file and feed-in the realistic values for the fluctuating sensors and other hardware events. As shown in Fig. 2(b), the STDNeut contains four core components: (i) config.ini generator, (ii) sensors manager, (iii) GPS manager, and (iv) GPS to BTS.

Config.ini Generator: It generates the `config.ini` file to spoof device-specific unique information.

Sensor Manager: It manages the device sensors by feeding-in realistic data periodically. To generate the value of sensors, it uses the same handles which are obtained through the Algorithm 1. The sensor manager manages all the sensors and other hardware events except the GPS. However, it gathers the next GPS coordinate to be projected by GPS manager so that the sensors on which GPS depends, can generate appropriate values.

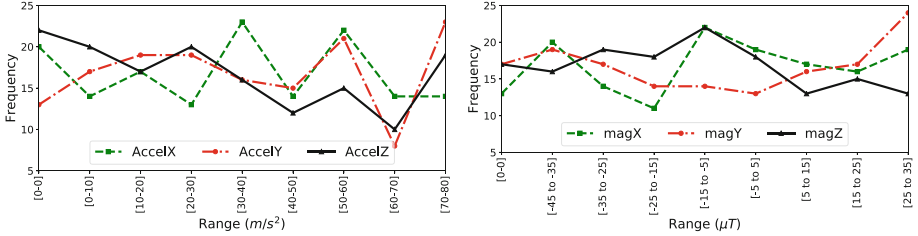
GPS Manager: The main reason behind the separate manager for the GPS is the correlation between the current GPS location and the previous location. For example, if the current GPS location is Washington DC, it is impossible for a person to reach New York in five minutes. Hence, a random GPS location alerts an App about the emulated environment. Therefore, a precise method is required to feed GPS location to an emulated environment, and GPS manager provides the same. The GPS manager reads the source and destination geo-location and the travel time from the STDNeut configuration file and generates a route by using a path patching algorithm, as shown in Algorithm 2. This algorithm takes source and destination geo-locations along with the number of steps required to move from source to destination, and returns the route trajectory.

GPS to BTS: A realistic GPS location alone is not strong enough to hide an emulated environment. It must be assisted by the BTS location that correlates with the current GPS location. This correlation is based on the maximum distance between the BTS and GPS locations, which may vary from 1 kms to 3 kms depending on the area’s population and obstacles. There are several commercial and public services that provide the GPS location by using a BTS ID. Still, no one provides the reverse mapping of it, i.e., providing a BTS ID based on GPS location and the SIM operator that is closer to the current GPS location. GPS to BTS module bridges this gap with the help of the OpenCellID database. The OpenCellID database contains information for the already installed BTS, worldwide, which is publicly available for research purposes. As this database stores BTS information worldwide, an efficient search mechanism is required to retrieve BTS ID based on the current GPS location and SIM operator. With this observation, we first filter the database based on the MCC, followed by the MNC. MCC and MNC reduce the search space to a specific operator within a country. Now we only need location area code and cell-ID to get the desired BTS ID, which is retrieved by calculating the distance with stored BTS location in the database and current GPS location, and compared against the maximum distance allowed. We have used `haversine` formula to measure the distance between the BTS location and current GPS location. The main reason for separate module for GPS to BTS correlation is because it requires to interact with an external database for retrieving the BTS ID according to the GPS location.

5 Validation of STDNeut

We use the Android Open Source Project (AOSP-7.1) to validate the proposed anti-emulation-detection system. For the experiments, Android Virtual Device (AVD) instances were configured with two CPU cores, 1.5 GB of RAM, 2 GB of internal storage and a 512MB of SD card along with all the sensors.

STDNeut vs. EmuDetLib: We evaluated the effectiveness of the STDNeut against EmuDetLib-Bench and RealMal samples (see Sect. 3.2). In evaluation, we found that STDNeut remains undetected against all the attacks performed by EmuDetLib-Bench and RealMal samples except the sample under category File info/SysProp and Mix of RealMal. The reason being the use of Qemu specific files and system properties that cannot be spoofed through the emulation layer. Hence, to bypass these detection methods, we have used the Xposed Framework. After evaluating the efficacy of the STDNeut, we attempt to understand this strong defense mechanism’s reasoning by performing various experiments. In the remaining part of this section, we discuss the reasons for the efficacy of STDNeut by analyzing different sensor readings and device information during the experiments. We also demonstrate a scenario for understanding the effectiveness of the STDNeut against distributed emulation-detection.



(a) Distribution of accelerometer readings with anti-emulation measures.

(b) Distribution of magneto-meter readings with anti-emulation measures.

Fig. 3. Effectiveness of STDNeut in neutralizing emulation detection using sensors by providing random reading for accelerometer and magnetometer.

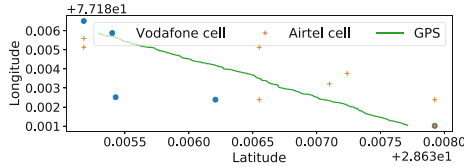


Fig. 4. GPS latitude and longitude reading with anti-emulation measures by feeding in realistic data along with associated BTS. GPS denotes path trajectory generated using the path patching algorithm.

5.1 Non-detectability Through Sensors

To evaluate the efficacy of STDNeut against potential malware exploiting sensor readings, we have developed an App to record and store the values of accelerometer, magnetometer, and GPS readings periodically which are shown in Fig. 3 and Fig. 4. In this evaluation, we have set two dependencies for sensors, one for time & GPS, and another for GPS & BTS. We make rest of the sensors as independent. The accelerometer reading represents the movement of the device in a three-dimensional space (referred to as AccelX, AccelY, and AccelZ) where the value in each dimension ranges from zero to ninety (0, 90). Fig. 3(a) shows the distribution of accelerometer readings where the X-axis represents ranges (total of nine ranges) of sensor values and the Y-axis represents the frequency. We have collected the values by executing an experiment for 150s and reading the sensor values every second. The data shows that all the sensor readings are almost equally likely and approximates a random distribution. Therefore, any emulation-detection technique based on accelerometer reading is nullified by our system. For the magnetometer (Fig. 3(b)), the magnetic field readings on each axis in a three-dimensional system are represented as magX, magY and magZ with a range between -45 to $+45$. As shown in the Fig. 3(b), the distribution is random, thus it does not allow an emulation-detection scheme using magnetometer data to succeed in detecting the underlying emulation platform.

Another source of emulation-detection is performed by reading GPS data. Unlike accelerometer and magnetometer, GPS data cannot be a random value. Depending on the location of the system, the GPS data should be provided with very slight variations in latitude and longitude. As shown in Fig. 4, STDNeut anti-emulation-detection measure can provide valid latitude and longitude values along with the associated BTS. In Fig. 4, GPS denotes the path trajectory generated using path patching Algorithm 2 whereas Vodafone cell and Airtel cell denotes the BTS location in the network of Vodafone and Airtel, respectively.

Table 3. Unique device information provided by STDNeut to three different AVDs executing simultaneously.

Queried Information	Information retrieved		
	AVD1	AVD2	AVD3
PhoneNumber	9876543210	9856543410	9876573213
IMSI	405541385237906	405521385237806	405511385238906
IMEI	359470010002931	359470010302943	359470010002949

5.2 Non-detectability Through Device Information

Device information is useful in differentiating between an emulated device and a real smartphone. In emulator platforms, device information such as IMEI, IMSI, phone number etc. are either absent or static values are present. To demonstrate the effectiveness of STDNeut’s anti-emulation-detection measures, we have used an App called `SIMCardInfo` [15], which extracts the information related to telephony services. We created three instances of this App in three different AVDs and executed all the instances simultaneously for one minute with and without STDNeut. The output of the App queries related to the device information is logged for all instances. We analyzed the log to extract information like IMEI and IMSI. Table 3 shows the captured device information with STDNeut. We are not showing the results other than the proposed system as the device readings were the same for all the instances. As shown in Table 3, STDNeut is capable of providing a unique device identity in a multi-instance setup. This is particularly useful to avoid detection when analyzing potential malware running in separate devices designed to operate in a collaborative manner as all malware see the same device identity. However, the values of PhoneNumber, IMEI, and IMSI are generated manually in the experiment which can be configured through the STDNeut’s configuration file without any modification in the Qemu.

5.3 Evading Distributed Emulation-Detection

To show the effectiveness of the STDNeut against emulation-detection using multiple clients along with a central server, we used Dendroid [28], a real Android

botnet. We integrated EmuDetLib into the Dendroid malware. We modified the Dendroid control server [28] not to send further instructions to the clients that seem to be running on emulated platforms by observing identical device information like IMEI from multiple clients (see Algorithm 2 at <https://skmtr1.github.io/EmuDetLib.html#a12>). Apart from hosting the control server, we also designed a victim site where the malware-infected devices perform a denial of service attack in a distributed manner when instructed from the control server. We created two instances for each of the CuckooDroid and STDNeut, and then executed Dendroid malware with integrated emulation-detection library. The control server instructs the infected devices to perform an HTTP flood on the victim site mentioned above only if the control server does not detect emulation. In our evaluation, we found that the control server is sending instructions only to the STDNeut system instances and not to the CuckooDroid instances. This was primarily because, the phone number, IMEI, etc. provided to the control server by the CuckooDroid were identical for both the instances, which was not the case with STDNeut. Therefore, we can conclude that the proposed STDNeut system can prevent emulation-detection orchestrated in a distributed setup.

5.4 Discussion and Limitations

Even though STDNeut provides a strong defense against all the malware samples, it falls short in the presence of malware that uses Qemu specific file and system properties for emulation-detection. To overcome this limitation, we have utilized the Xposed framework. The Xposed framework itself is susceptible of detection from App. For example, the Snapchat App uses the native code to detect Xposed [2]. It is possible because Xposed capability is limited to the framework level API only, and here detection is performed through the native code. A more suitable defense is to use kernel-level modification that remains undetected even when the attack is performed from any layer above the kernel. However, our malware set does not contain any samples that detect the existence of Xposed.

Additionally, the first eight digits of IMEI are called TAC (Type Allocation code), which indicate the device type. Malware can also use TAC to detect an emulated environment by observing TAC's mismatch with the Android device name. To our knowledge, we have not observed the existence of such malware. However, STDNeut is a generic solution that requires analyst intervention to configure it with appropriate information like selecting device type and corresponding TAC value in IMEI and other such information.

Furthermore, some Android devices like tablets may lack cellular capabilities or do not have some sensors like GPS. In STDNeut, cellular, GPS, and other sensors fall under the sensor category. An analyst may configure STDNeut without these sensors information to create a realistic emulated device where such sensors are not present.

Evaluation Summary: In a nutshell, the proposed STDNeut can effectively execute a malware without being detected as an emulated environment.

6 Conclusion

This paper proposed a flexible and configurable emulation-detection library (EmuDetLib) that provides extensive emulation-detection methods. We have used EmuDetLib to show that anti-emulation-detection measures of the existing dynamic analysis frameworks are not sufficient. Moreover, beyond basic defense against emulation-detection, all the analysis frameworks fail to hide the underlying emulation layer. To design a robust analysis framework on emulated platforms, we proposed STDNeut, a configurable anti-emulation-detection system. STDNeut hides the emulated platform effectively by handling the data from sensors, telephony system, and device attributes in a realistic manner. We performed experiments to demonstrate the effectiveness of STDNeut against the primary and extended detection methods. We believe STDNeut provides efficient and secure anti-emulation-detection measures that are difficult to be bypassed even by sophisticated malware.

Acknowledgements. We thank our shepherd Matthias Wählisch and all the anonymous reviewers for their helpful comments and suggestions. This work is supported by Visvesvaraya Ph.D. Fellowship grant MEITY-PHD-999.

References

1. AT Commands - 3GPP TS 27.007 (2020). <https://doc.qt.io/archives/extended4.4/atcommands.html>
2. AeonLucid: Snapchat detection on Android - Aeonlucid (2019). <https://aeonlucid.com/Snapchat-detection-on-Android/>
3. AG, G.S.: A new malware every 7 seconds (2019). <https://www.gdatasoftware.com/news/2018/07/30950-a-new-malware-every-7-seconds>
4. Allix, K. et al.: Androzoo: collecting millions of android apps for the research community. In: MSR, pp. 468–471 (2016)
5. Android Developers: Run apps on the android emulator — android developers (2019). <https://developer.android.com/studio/run/emulator>
6. Android Developers: Send emulator console commands — Android developers (2019). <https://developer.android.com/studio/run/emulator-console>
7. Arakawa, Y.: Emulatordetector: Android emulator detector unity compatible (2019). <https://github.com/mofneko/EmulatorDetector>
8. Arzt et al.: Droidbench 3.0 (2019). <https://github.com/secure-software-engineering/DroidBench/tree/develop>
9. Bellard, F.: Qemu, a fast and portable dynamic translator. In: ATEC, p. 41 (2005)
10. Costamagna, V. et al.: Identifying and evading android sandbox through usage-profile based fingerprints. In: RESEC (2018)
11. Desnos et al.: Welcome to Androguard’s documentation! - androguard 3.3.5 documentation (2019). <https://androguard.readthedocs.io/en/latest/>
12. Diao, W. et al.: Evading android runtime analysis through detecting programmed interactions. In: WiSec, pp. 159–164 (2016)
13. Fenton, C.: Android emulator detect — calebfento (2019). <https://github.com/CalebFenton/AndroidEmulatorDetect>

14. Gingo: Android-emulator-detector: Small utility for detecting if your app is running on emulator, or real device (2019). <https://github.com/gingo/android-emulator-detector>
15. Gonzalez, H.: Sim card info - apps on google play (2019). https://play.google.com/store/apps/details?id=me.harrygonzalez.simcardinfo&hl=en_IN
16. IDC: IDC-smartphone market share - OS (2019). <https://www.idc.com/promo/smartphone-market-share/os>
17. Inc., F.: Android emulator detector: Easy to detect android emulator (2019). <https://github.com/framgia/android-emulator-detector>
18. Jing, Y. et al.: Morpheus: Automatically generating heuristics to detect android emulators. In: ACSAC, pp. 216–225 (2014)
19. Kudrenko, D.: Emulator-detector: Detect emulators like genymotion and Nox player by accelerometer (2019). <https://github.com/dmitrikudrenko/Emulator-Detector>
20. Lab, A.: Argus SAF - argus-pag (2019). <http://pag.arguslab.org/argus-saf>
21. Lantz, P.: An Android Application Sandbox for Dynamic Analysis. Master's thesis (November 2011) <https://www.eit.lth.se/sprapport.php?uid=595>
22. Lockheimer, H.: Android and security - official google mobile blog (2012). <http://googlemobile.blogspot.com/2012/02/android-and-security.html>
23. Maruyama, S., et al.: Base transceiver station for w-cdma system. Fujitsu Sci. Tech. J. **38**, 167–173 (2002)
24. MobSF Team: 1. documentation. MobSF/mobile-security-framework-MobSF wiki (2019). <https://github.com/MobSF/Mobile-Security-Framework-MobSF/wiki/1-Documentation>
25. Oberheide, J., Miller, C.: Dissecting the android bouncer (2012). <https://jon.oberheide.org/files/summercon12-bouncer.pdf>
26. Orłowski, A.: Google play store spews malware onto 9 million 'Droids. the register (2019). https://www.theregister.co.uk/2019/01/09/google_play_store_malware_onto_9m_droids/
27. Percoco, N.J., Schulte, S.: Adventures in BouncerLand (2012). https://media.blackhat.com/bh-us-12/Briefings/Percoco/BH_US_12_Percoco_Adventures_in_Bouncerland_WP.pdf
28. Qqshow: Github - qqshow/dendroid: Dendroid source code. contains panel and Apk. (2019). <https://github.com/qqshow/dendroid>
29. Rasthofer, S. et al.: Harvesting runtime values in android applications that feature anti-analysis techniques. In: NDSS (2016)
30. Sadeghi, A., et al.: A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software. IEEE Trans. Softw. Eng. **43**(6), 492–530 (2017)
31. Sun, M. et al.: TaintART: a practical multi-level information-flow tracking system for android runtime. In: ACM SIGSAC CCS, pp. 331–342 (2016)
32. Tam, K. et al.: Copperdroid: automatic reconstruction of android malware behaviors. In: NDSS (2015)
33. Tam, K., et al.: The evolution of android malware and android analysis techniques. ACM Comput. Surv. **49**(4), 76:1–76:41 (2017)
34. Technologies, C.S.: CuckooDroid book (2014). <https://cuckoo-droid.readthedocs.io/en/latest/>
35. thehackernews.com: New android malware apps use motion sensor to evade detection (2019). <https://thehackernews.com/2019/01/android-malware-play-store.html>

36. Vidas, T., Christin, N.: Evading Android runtime analysis via sandbox detection. In: ASIA CCS (2014)
37. Wang, X. et al.: Droid-AntiRM: taming control flow anti-analysis to support automated dynamic analysis of android malware. In: ACSAC (2017)
38. Wei, F. et al.: Amandroid: a precise and general inter-component data flow analysis framework for security vetting of android apps. In: ACM SIGSAC CCS (2014)
39. XDA Developers: Xposed framework hub (2019). <https://www.xda-developers.com/xposed-framework-hub/>
40. Yan, L.K., Yin, H.: Droidscape: seamlessly reconstructing the OS and dalvik semantic views for dynamic android malware analysis. In: USENIX Security (2012)



HMAC and “Secure Preferences”: Revisiting Chromium-Based Browsers Security

Pablo Picazo-Sanchez^(✉), Gerardo Schneider, and Andrei Sabelfeld

Chalmers University of Technology, Gothenburg, Sweden
Pablop@chalmers.se

Abstract. Google disabled years ago the possibility to freely modify some internal configuration parameters, so options like silently (un)install browser extensions, changing the home page or the search engine were banned. This capability was as simple as adding/removing some lines from a plain text file called Secure Preferences file automatically created by Chromium the first time it was launched. Concretely, Google introduced a security mechanism based on a cryptographic algorithm named Hash-based Message Authentication Code (HMAC) to avoid users and applications other than the browser modifying the Secure Preferences file. This paper demonstrates that it is possible to perform browser hijacking, browser extension fingerprinting, and remote code execution attacks as well as silent browser extensions (un)installation by coding a platform-independent proof-of-concept changeware that exploits the HMAC, allowing for free modification of the Secure Preferences file. Last but not least, we analyze the security of the four most important Chromium-based browsers: Brave, Chrome, Microsoft Edge, and Opera, concluding that all of them suffer from the same security pitfall.

Keywords: HMAC · Changeware · Chromium · Web security

1 Introduction

Chrome is as of today the most used web browser in the world [42]. Chrome, as well as many other browser vendors like Opera, Brave and Vivaldi are based on Chromium, an open-sourced web browser developed by Google. Recently, Microsoft moved to adopt Chromium as the basis for the new Microsoft Edge browser [27]. Given its widespread use, around 75% of the desktop users on Internet [38], the security of Chromium is paramount.

To allow easy customization of the web browser to fit the needs of the users, many configuration parameters may be modified. Setting the homepage to a custom webpage a user frequently visits, changing the default search engine, “pinning” some URLs to tabs and browser extensions management, are just a few examples of the huge list of actions that can be performed to make the user

experience more pleasant. One of the most promising tools for enriching the browser experience of the user is *browser extensions*. Extensions are installed from the Chrome Web Store, which is a central repository managed by Google.

As recently claimed [18], approximately 10% of the browser extensions stored between 2012 and 2015 in the Web Store were classified as malware and deleted from the repository. Despite many attempts done to improve the security and privacy of extensions [18, 19, 34, 36], vulnerabilities still abound [2, 3, 35], being PUPs one popular and challenging example because they are not usually marked as malware by antivirus vendors [21, 40].

PUPs are installation executable files that, apart from installing the application the user wants, they also execute other software that might not be related to the legitimate one. *Adware* and *changeware* are two types of PUPs that add advertisement to the webpages the user visits and changes the configuration properties of the browser silently, respectively. Recently, a cybersecurity firm discussed the thin line between espionage-level malware and PUPs and detected more than 111 browser extensions considered to be PUP whose goal was to spy users [4]. In this paper, we consider PUPs and pay special attention to how changeware works, providing a concrete example of how the installation of *uTorrent* application modifies the configuration of the browser (see Sect. 3).

In the particular case of Chromium-based browsers, each user obtains a couple of configuration files for storing information such as bookmarks, history, homepage and other preferences. One of these files is the *Secure Preference file* which is automatically loaded when the browser is launched and it is updated each time the browser is closed. In 2012 Google improved its browser's security to protect users from silently installing extensions since these were causing more and more problems. Before that, it was possible to silently install extensions into Chrome by directly modifying the Secure Preferences file or by using the Windows registry mechanism. Extensions that were installed by third party-programs through external extension deployment options were disabled by default and only extensions installed from Google Web Store are now allowed.

Concretely, from its version 25 Chromium implemented a security mechanism to ensure that no external applications apart from the browser can modify the Secure Preferences file. This mechanism is a custom Hash-based Message Authentication Code (HMAC) algorithm [22] which produces a SHA-256 hash given both a seed and a message. However, as the original authors claimed, the security of HMAC relies on the seed generation, thus being secure as long as the seed is.

Our findings reveal that the seed needed to generate the HMAC, stored in a public file named `resources.pak`, is not randomly generated. Moreover, for each Chromium-based browser, the seed is the same for all the OSs. Nevertheless, if the seed were randomly generated the problem of where to securely store either the seed or the key used to encrypt the seed, still persists. In previous work, it has been proposed to use WhiteBox-Cryptography [10] to secure this seed on Chromium [6]. However, this solution is platform-dependent, and only works under certain circumstances and on a concrete OS. As we show in this paper

the problem remains unsolved. Once a malicious party gets such a seed, it may impersonate the browser and modify any parameter of the Secure Preferences file.

To the best of our knowledge, the attack against the Secure Preferences file has never been published with the exception of a partial description in (at least) one Internet forum—whose moderator claimed that this attack no longer works [17]. To confirm this, we downloaded and installed multiple versions of Chromium in computers with Windows 10 and MacOS. We implemented the attack described in that forum and confirmed that it did stop working from Chromium versions up to 58.0.2999.0. In this paper, we present a proof-of-concept PUP that modifies the Secure Preferences file of any Chromium version from 58.0.2999.0 until the latest one at the time of writing (85.0.4172.0). Additionally, if used together with the attack presented in that forum, any Chromium version can be easily editable (see Table 1).

Table 1. Chromium versions exploitable via HMAC.

Chromium Version	Released	SPF
(prior to) 25.0.1313.0	2012	Free modification
25.0.1313.0	2012	Attack [17]
58.0.2988.0	2017-01	Attack [17]
58.0.2999.0	2017-02	This paper
85.0.4172.0 (latest)	2020	This paper

This poses serious security and privacy issues. For instance, it is possible to perform browser hijacking attacks [31, 43], fingerprinting attacks [2, 23, 34], remote code execution [35], as well as silent browser extensions (un)installation (something Google has in principle banned years ago [11]). In many cases, the way of proceeding is the same: changing the browser search provider to generate advertising revenue by using well-known search providers like Yahoo Search or Softonic Web Search among others [1, 25]; retrieving information about that uniquely identifies the user, and; exploiting other extensions to gain privileges or to remotely execute source code.

Contributions. This paper analyzes how four of the most important Chromium-based browsers [15]—Chrome (70% of market share), Microsoft Edge (5% of market share), Opera (2.4%), and Brave¹—manage the security and privacy of the users through a configuration file named Secure Preferences file. We discover that all of them use fixed seeds to generate the HMACs to secure the Secure Preferences file. These HMACs are used to guarantee that the content of the users’ privacy settings has not been altered by any other party different than the browser (Sect. 2.2). We implement a changeware that impersonates the browser

¹ Brave uses Chrome user-agent (desktop and Android) and Firefox user-agent (iOS).

and (un)install extensions, perform phishing attacks, hijack the user’s browser, fingerprint users through the extensions the browser has, among other things (Sect. 3).

Section 2 presents background information concerning the Secure Preferences file and how Chromium uses it. Section 4 exposes some countermeasures to avoid the attack as well as a brief discussion about how this vulnerability can be used by the research community for analyzing browser extensions. Finally, Sect. 5 presents the related work and Sect. 6 concludes the paper.

2 Background

In this section, we explain the role of the Secure Preferences file and how the HMAC is generated in Chromium-based browsers.

2.1 Chromium Preferences

To manage and enforce configurable settings, Chromium implements a mechanism called *preferences* to modify the settings of the browser per user instead of doing this centrally. Using preferences it is possible to configure, for instance, the homepage, which extensions are enabled/disabled and the default search engine.

We show how Secure Preferences file works via an example. Let Alice be a user who wants to manually modify any of the preferences stored in the Secure Preferences file. She accesses her profile’s folder, opens the JSON file—all the preferences are stored in plain text so anyone can access that file—and manually alters the preferences she wants to. Once she has modified the file, she saves it and launches her Chromium instance to check whether the changes have been applied or not. When Chromium loads, it automatically checks the integrity of the Secure Preferences file, warning Alice that the file has been externally modified and the browser marks the file as corrupted. Chromium then automatically restores the Secure Preferences file to either a default or to a previous safe state.

Alice, who is an advanced user, tries to cheat Chromium by launching the web browser and manually modifying the Secure Preferences file when the browser is running expecting her changes to take effect. That, however, will not work since Chromium loads the Secure Preferences file when it is launched the first time and overrides the whole Secure Preferences file when Alice closes the browser.

The rationale behind Chromium’s behavior is to avoid external modifications to the Secure Preferences file for privacy reasons. In particular, what makes the Secure Preferences file secure is that Google added a Hash-based Message Authentication Code (HMAC) signature of every entry (*settings/preference*) in the file. In addition to this, the file also has a global-HMAC called *super_mac* to check the integrity of all the other HMACs.

HMAC [22] is a particular case of (MAC) which involves a hash function in combination with a shared secret key—also known as *seed* in these schemes. This algorithm was created in the 90’s and is usually used for both data verification and message authentication. As stated in the original proposal, the security of

the HMAC protocols rely on the security of the underlying hash function, as well as both the size and quality of the seed.

Finally, if all the HMACs of the Secure Preferences file are correct, the browser will set up the settings according to what is stated in that file. In the case the validation procedure fails, the browser will use the default values for those where the HMAC validation failed. This recovery process is the same for all the Chromium-based browser but Brave. In this particular browser, instead of restoring the file to a previous state, it keeps a copy in the file system of the “corrupted” preferences file (using `.old` extension) and creates a new one.

2.2 HMAC in Chromium

From version 25.0.1212.0 released in 2012, Google decided to not allow other parties different than the browser to modify the user’s settings by including an HMAC per setting stored in the Secure Preferences file. When the user closes the browser, it computes the HMAC whereas when the user opens it, the browser recomputes all the HMACs and checks whether they were created by the browser. In particular, to modify the Secure Preferences file, the browser needs to: a) acquire the seed, and b) obtain the message. Once the browser has these data it computes both the HMACs of the settings, and a final HMAC called *super_mac*.

Acquiring the Seed. The seed is stored in the `resource.pak` file. We explain in what follows how we get the seeds of the latest versions as of June 2020 of the four browsers being considered.

Chrome. The seed that Chrome uses to compute the HMAC is a 64-long character hexadecimal string that can be found in the `resource.pak` file. Concretely, the first resource that has a length of 256 binary bits in the `resource.pak` file is the seed Chromium uses. Roughly speaking, we obtain this resource by loading the file and seeking for the first line (`resource`) with 64 characters.

Table 2. Seed calculation on different OS

OS	#PC	Same seed
Linux	48	✓
Windows	44	✓
MacOS	8	✓

We executed the script on 100 different computers with different OSs (48 Linux, 44 Windows and 8 MacOS) and the results can be seen in Table 2. Concluding that the seed is not randomly computed as claimed. Concretely, the seed is: `b'\xe7H\xf36\xd8^\xa5\xf9\xdc\xdf%\xd8\xf3G\xa6[L\xdfv\x00\xf0-\xf6rJ*\xf1\xa!-&\xb7\x88\xa2P\x86\x91\x0c\xf3\xa9\x03\x13ihq\xf3\dc\x05\x8270\xc9\x1d\xf8\xba\0\xd9\xc8\x84\xb5\x05\xa8'`. We run this experiment on Chrome version 85.0.4172.0.

Brave, Microsoft Edge and Opera. We executed the same script as for Chrome to extract the seed on Brave, Edge and Opera but we could not change the user’s settings. We had then to perform a brute force attack to extract the seed because the file was different than in Chrome. We got an alarming result concerning these four vendors: the seed is the blank string, i.e., `seed = b''` in both Windows and MacOS. The version of Microsoft Edge we used was 85.0.564.51,

for Brave we used version 1.14.81 (based on Chromium: 85.0.4183.102) whereas for Opera we used version 71.0.3770.148.

Obtaining the Message. To correctly generate the HMAC, a message should be passed as input. This message is composed of a *MachineIdStatus* and a string message. Such a variable is platform-dependent, i.e., the *MachineIdStatus* is a different value in Windows, Linux and MacOS. That said, all four browsers have similar procedures to create the message used to generate the HMAC. In what follows we detail how the three different platforms obtain that *MachineIdStatus* value.

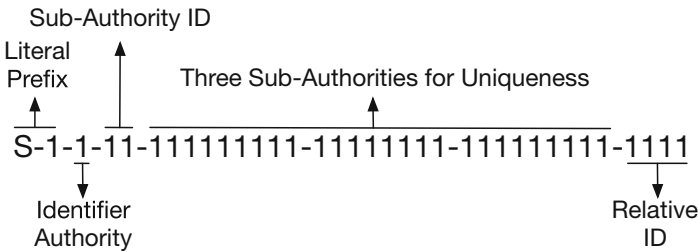


Fig. 1. Security Identifier (SID)

Windows. Users are provided with a unique identifier named SID. This identifier is usually used to control the access to resources like files, registry keys and network shares, among others. An example of the SID can be seen in Figure 1 and it might be easily retrieved by executing either the `wmic` or the `whoami` commands on Windows. After retrieving the SID, the last characters (Relative ID in Figure 1) are deleted for the final usage.

MacOS. Instead of using the SID, MacOS uses the hardware (UUID) which is a 128-bits number obtained by using the command `system_profiler SPHardwareDataType`. It outputs a hexadecimal number split in five groups by a “-”, e.g., 1098AB78-6BF1-517E-905A-F018AABC4B26. In particular, in the `device_id_mac.cc` we can find how Chromium retrieves that UUID which is used afterwards as part of the message.

Linux. Both Windows and MacOS have their own files under `chromium/src/services/preferences/tracked/` directory but there is no references about Linux. We corroborate that by checking the `device_is_unittest.cc` file where we found an if-then-else statement to differentiate how the SID should be computed depending whether the OS is either Windows or Mac OS but there are no rules for Linux. Consequently, when the browser is running on Linux, the else statement is executed where there is a `MachineIdStatus::NOT_IMPLEMENTED;`. As a consequence, the *MachineIdStatus* variable has an empty string.

By manually analyzing the message in Chromium, we realized that it is composed of key of the Secure Preferences file value it wants to modify together with either the SID (or the UUID) of the current user (or computer). More concretely, Chromium implements a function named `GetMessage` in the `pref_hash_calculator.cc` file, whose purpose is to concatenate three parameters given as inputs: `Device_ID`, `path` and `value`.

`Device_ID` corresponds to the `MachineIdStatus`, i.e., UUID on MacOS or the SID of the user without the relative ID information on Windows or the empty string on Linux. In other words, `Device_ID` is the identifier of the machine where Chromium is installed. Since every machine has its own unique SID no two HMACs will be the same when computed on different machines. However, on MacOS, since that the UUID is linked to the machine instead of being associated to the user, different profiles in the same machine will have the same UUID value.

`Path` is where the Secure Preferences file is in the computer. It has a concrete format that uses dots (“.”) as delimiters. For example, the preference that handles if the home button is visible or not is `show_home_button`, being the path `browser.show_home_button` and it contains a Boolean value.

The final HMAC is a string where all empty arrays and objects are removed and the character “!” is replaced by its Unicode representation (“\u003C”). In the example, the value of the home button would be `"show_home_button":true`.

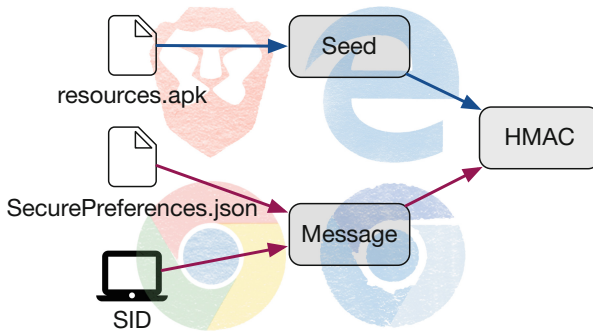


Fig. 2. HMAC protocol in Chromium based browsers

HMAC Reproduction. The function `GetDigestString`, located in `pref_hash_calculator.cc` file, is the one that generates an HMAC given a message and a key as inputs. The key and the message are as described above. We can impersonate the browser and generate HMACs to change any of the values of the Secure Preferences file as if we were the browser. An illustrative summary of the HMAC protocol in Chromium-based browsers can be seen in Fig. 2. Once the HMACs are computed (one per modified value in the Secure Preferences file) they are then combined to create a new message that is used as input of the hash algorithm to calculate the final HMAC called *super_mac*.

The Secure Preferences file is then updated with the result of these calculations together with the modified preference values.

Chromium has a validation mechanism to check the integrity of the HMACs which is also calculated in the `Validate` function of the `pref_hash_calculator.cc` file. Such a function takes three parameters as input: a path of the JSON file, a value of the JSON file and a digest string which is the current HMAC of that value. Inside that function, another function called `VerifyDigestString` (which is also located in the same file, i.e., `pref_hash_calculator.cc`) takes as inputs a key (a string), a message (generated from the function `GetMessage` on `pref_hash_calculator.cc`), and a digest string (the HMAC). After being verified by the function `Verify` located on `hmac.cc`, a SHA256 string is returned.

3 Security Analysis

In what follows, we introduce the attacker model and provide some examples that exploit the HMAC detailed in the previous section to modify the Secure Preferences file. We present a proof-of-concept whose source code we released for future research on the field². Finally, we analyze the main differences between the installed-by-default extensions in Brave, Chrome and Edge and Opera browsers, and demonstrate how an external server can execute some parts of the code of the installed-by-default extensions creating a big security threat.

3.1 Attacker Model

Our attacker model is composed of any software application that specifically alters the Secure Preferences file of Chromium. This attacker model is known in the literature as PUPs which are executable files that apart from the desired program installation also install other software that might not be related to the legitimate one, typically *adware* [21,40]. What makes PUPs different from malware is that users are tricked to approve the installation of this third party application. Typically, during the installation process the PUPs shows a message that the user has to (un)check before the process continues.

More specifically, there is a subset of PUPs called *changeware* whose aim is to modify the settings of the browser [6], usually for malicious purposes as confused deputy. Let us give an illustrative example. Years ago, Oracle used to include in the Java installation file one selected-by-default checkbox by which a Yahoo toolbar was automatically installed unless the user did not manually uncheck it during the installation process [39]. Similar cases were seen with WinYahoo which was installed as part of the Adobe Photoshop Album Starter Edition software [26]. In all the aforementioned cases, the attackers are the binaries that modify the Secure Preferences file. In both cases, unwanted browser extensions are installed in the user's browser. Note that browser extensions can usually have access to any website (sensitive or not) that the user visits.

² <https://github.com/Pica4x6/SecurePreferencesFile>.

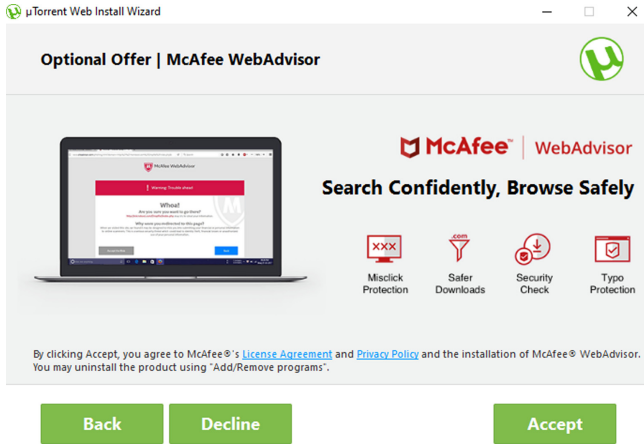


Fig. 3. uTorrent installation process.

Even though the issue made apparent in the examples above was identified and marked as PUPs some time after its detection, there still are many up-to-date examples of applications where browser extensions are piggyback programs. One such example is *uTorrent Web* binary installation. Concretely, during the installation process, the user has to accept or decline the installation of *McAfee WebAdvisor* software (see Fig. 3). If the user accepts, that “extra” software is installed together with a browser extension which is automatically installed in Chrome (see Fig. reffig:McAfeeExtension). However, when the user manually uninstalls the McAfee WebAdvisor application, the extension might also be uninstalled from Chrome. This clearly indicates that there still are applications that can install browser extensions without requiring the user to use Google WebStore as it is claimed.

3.2 Changeware Proof-of-Concept

The challenging part of PUPs is that they are not usually marked by antivirus vendors as malicious software [21, 40]. Windows claimed to stop PUPs and they even added such an option as part of the recently released Microsoft Edge [12]. We created a changeware and confirmed that it is not classified as malware by the Microsoft detection mechanism [12, 29]. Additionally, we run a set of popular online antivirus tests like Virustotal³, MetaDefender⁴ and VirScan⁵ and our changeware passed all the security checks. As a conclusion, we can effectively alter the Secure Preferences file with no restrictions at all. All files to reproduce the attacks above are publicly available in <https://github.com/Pica4x6/SecurePreferencesFile>.

³ <https://www.virustotal.com>.

⁴ <https://metadefender.opswat.com>.

⁵ <https://www.virscan.org>.

3.3 Practical Attacks

We analyze now the main attacks a changeware can exploit. In particular, we classify the attacks into browser hijacking and browser extensions. Most of these attacks are interconnected since the goal of the attacks is to get the private information of the user. Some years ago Kotzias *et al.* [21] analyzed almost 4 million hosts and conclude that half of them have some kind of PUPs installed. More recently, Urban *et al.* [40] analyzed the communication carried out by 16k PUPs and 5.5k Firefox extensions and got that almost 40% and 45% include personal information of the user respectively.

Browser Hijacking. The goal of this attack is to increase the advertising revenue by forcing the user to access concrete webpages. To redirect users to such sites, changeware may modify up to five main values of the Secure Preferences file, namely: [i)] `homepage`; [ii)] `pinned_tabs`; [iii)] `import_bookmarks_from_file`; [iv)] `search_engine`, and; [v)] `sessions` keys. Antivirus vendors usually identify this attack by parsing the Secure Preferences file and analyzing the URLs defined in it. If they belong to a *blacklist* the antivirus constantly keeps updated, then an unwanted change might be detected and the antivirus analyzes the disk looking for malicious software. However, this method can be bypassed by modifying the `import_bookmarks_from_file`. This is a special option in the manifest which states the path where Chrome silently and automatically imports bookmarks from the HTML stated in the path of such key.

Phishing The goal of this type of attack is to steal user's private information. This is done by loading a fake webpage that looks similar to the legitimate one. If not aware of the URL, the user will interact with the site as usual. To trick users, browser extensions can implement some strategies to redirect them to fake pages and perform phishing attacks [41], analyze the most visited web pages and generate bookmarks, change the pinned tabs they already have or even generate new ones.

Browser Extensions: Execution Order, Paths and Fingerprinting Recently, Picazo-Sanchez *et al.* demonstrated that the order in which browser extensions are executed may alter the content of the DOM and the behavior of the browser in general [30]. The attack was implemented corroborating that the changeware could modify the installation time of extensions altering the execution order. Moreover, it can also modify any of the paths the extensions define in the manifest, being possible to include new paths in the user's file system loading different extension files. Different techniques have been proposed so far to fingerprint browser extensions, i.e., using WARs [34], using behavioral-based enumeration [36,37] or because inter or extra communication messages [20,35]. Any of these methods can be easily exploited by the changeware. This could be done for instance by defining and including new resources as WARs, including JavaScripts into the extensions files that automatically inserts content into the DOM, deleting the `externally_connectable` key of the extensions so that other

webpages can send messages to the background pages of the extensions (we show an example of this attack in Sect. 3.4 by analyzing the installed-by-default extensions) or a combination of them.

Browser Extensions: Permissions Chromium offers a set of APIs that extensions can use, being some of them accessible by defining the corresponding permission in the manifest of the extensions. Once installed, the manifest is parsed and stored as part of the Secure Preferences file under the extension id key, therefore the original manifest is no longer checked. This poses serious security issues since any changeware might alter the permissions the user agreed upon and either provide it with more or fewer permissions than initially. Let us give a concrete example, the browser extension whose id is `mgpdmkhhjffhfbpeigghejkngiaaike` and more than 400,000 downloads, includes in the `background.js` file `return chrome.webRequest.onCompleted` but it does not include the `webRequest` permission needed to execute such statement. The changeware can easily add such permission into the Secure Preferences file giving access to that API to the extensions and thus, executing that line without any errors.

Browser Extensions: Silent Installation Even though Google banned silent browser installations in 2012, we managed to successfully install, delete, activate and deactivate browser extensions. See Fig. 4 for a real example used today by software that includes a browser extension in the browser without using the official WebStore. In this particular case, the extension can be seen and manually removed by the user, but this might not always be the case. In the worst-case scenario, the changeware could have the source of the extension to be installed inside the binary file.

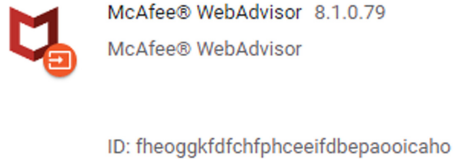


Fig. 4. McAfee browser extension.

Note that is then possible to install new browser extensions without the user being notified similarly to how installed-by-default extensions work (see Tables 5 to 6). Concretely, enabling and disabling extensions is determined by the `state` key of the extension in the Secure Preferences file being straightforward to modify them. Also, remark that to uninstall an extension, the changeware can simply delete the whole entry of the preferences file and computing the `super_mac` of the Secure Preferences file.

3.4 Installed-by-Default Extensions

In the following, we analyze the extensions that are installed by default with the browser, for all four browser under consideration.

Brave. This browser has ten installed-by-default apps where none of them are extensions and they cannot be removed by the user (see Table 3). Brave, renames some of the default extensions despite being exactly the same as the

one provided by Chromium, e.g., *Chromium PDF viewer*. However, what makes Brave different from the other browsers is that it allows the user to disable (not to uninstall) some installed-by-default apps, e.g., WebTorrent, Google Hangouts and Crypto Wallet.

Chrome. We confirmed that the number of default extensions is sixteen (between browser extensions and apps) in the three OSs. In addition to that, only a subset of them might be uninstalled by the user in the usual way, i.e., going either to `chrome://extensions` or `chrome://apps` and manually removing them. We show a detailed list of the installed-by-default extensions in Table 5.

Regarding the platform, our initial hypothesis was that Linux was the most privacy compliant of the evaluated OSs. After running the first part of the experiments we confirmed that indeed Chrome does not install any single browser extension by default in Linux (contrarily to what happens in other OSs), and the file is totally empty except for the `super_mac` key. We realized that Linux does not modify such a file but **Preferences** file, so we had to adapt our tests to use that Preferences file instead.

Microsoft Edge. Unlike Chrome, Microsoft Edge has ten installed-by-default applications and only one extension (see Table 4). Most, if not all, browser extensions developed for one particular Chromium-based browser can be easily exported to other Chromium-based browsers. There are cases where vendors can modify some parameters like the name of the extensions, e.g., `mhjfbmdgcfjbbpaeojofohoefgihjai` is named here *Microsoft Edge PDF Viewer* and *Chrome PDF Viewer* on Chrome. We tested Edge on Windows and MacOS without noticing differences when the Secure Preferences file is generated for the first time. It is also interesting to mention that there is no way for the user to get rid of any default extensions on Edge.

Opera. This browser is the one that has more information by default in the Secure Preferences file when it is installed for the first time. Concretely, there are more than 300 extensions hardcoded whose purpose is to ban them to be installed by the user— a disallowed list. Other than that, there are 21 extensions installed by default and none of them can actually be uninstalled nor disabled by the user. Table 6 shows a list of the installed-by-default extensions in Opera.

3.5 Google Hangouts Use Case

There is an unexplored set of extensions that are typically overlooked by the research community: installed-by-default extensions. We manually analyzed all of them and realized that *Google Docs Offline*, *Chrome Media Router*, *CryptoTokenExtension* and *Google Hangouts*, implement external message listeners and have the `externally_connectable` key defined in their manifest files. Given that only the *Google Docs Offline* extension can be deleted by the user, if a changeware modifies such key in the Secure Preferences file any website can send messages to these extensions as if they were legitimate websites.

Concretely, Google Hangouts is one of these installed-by-default extensions present in all the Chromium-based browsers. It cannot be uninstalled by the

user with the only exception of Brave which can be disabled. In the following, we analyze it to demonstrate the information that an attacker can get by exploiting the Secure Preferences file.

A recent study performed by Somé demonstrates how browser extensions allow web applications to bypass the Same Origin Policy and have access to sensitive information of the user [35]. Concretely, extensions that listen for external messages should be defined in advance in the manifest similar to what the `externally_connectable` does. If, for instance, such a key is not defined in the manifest file and extensions implement any of the external message listeners (i.e., `onMessageExternal` and `onConnectExternal`), they will listen and execute the code defined in any of these functions. Moreover, if any dangerous function like `eval()` is defined in these listeners, the consequences might be catastrophic since the attacker can take control of the extension and run arbitrary code on it.

```
var editorExtensionId="nkeimhogjdpnpccoofpliimaahmaaome";
var port_a = chrome.runtime.connect(editorExtensionId,{name: "
processCpu"});
var port_b = chrome.runtime.connect(editorExtensionId,{name: "
chooseDesktopMedia"});
port_b.postMessage({
method: 'chooseDesktopMedia',
// sources: ['screen','window','tab','audio']
sources: ["window"]
});

port_a.onMessage.addListener(
function(msg) {console.log(msg)});
port_b.onMessage.addListener(
function(msg) {console.log(msg)});
```

Fig. 5. Script that the attacker injects to extract information from the user.

Google Hangouts defines the pattern `https://*.google.com/*` as trusted webpages, making thus possible for any (sub)domain of `google.com` send messages to the extension. We created a dummy server (`http://www.attacker.com`) and added it to the `externally_connectable` list by using the attack described in Sect. 3. We manually analyzed such extension and realized that the attacker can obtain information like `{"browserCpuUsage":2.2,"gpuCpuUsage":3.0,"tabCpuUsage":0.0,"tabJsMemoryAllocated":3133440,"tabJsMemoryUsed":1743032,"tabNetworkUsage":0}` by using a port named “processCpu” (line 3 of Fig. 5).

Apart from that, by setting the right parameters (lines 4 and 5 of Fig. 5) the attacker may execute the `chrome.desktopCapture.chooseDesktopMedia(array of DesktopCaptureSourceType sources, tabs.Tab targetTab, function callback)` function where the array of sources can be either “screen”, “window”, “tab”, or “audio” according to the official documentation provided by Google. The attacker can then get a screenshot of: 1) the current screen of the user’s computer; 2) any software the user is running; 3) the tab of the browser, and; 4) the audio of user.

Finally, with our changeware we can remove the entire `externally_connectable` entry of the Hangouts extension from the Secure Preferences file. In fact, any browser extension can execute those functions and retrieve such information. In addition, by including a list of allowed sites in the `matches` list of the `externally_connectable` key, any website can also execute and get these data.

4 Discussion

Here we discuss countermeasures and proposals to prevent the Secure Preferences file attack as well as the potential benefits that our attacks have for future analysis of the extensions.

Coordinated Disclosure. We contacted the four vendors: Brave, Google, Microsoft and Opera to report our findings. Brave is in progress of fixing the problem. Google acknowledged that “defeating the HMAC is a signal that the software is in violation of the Unwanted Software Policy”. Both Microsoft and Opera deferred to the Chromium project.

Preventing the SPF Attack. In our approach, we first generate a nonce—a random values of 64-character long string—from a uniform distribution, and use it to replace the seed already stored in the `resources.pak` file. When we launched Chromium for the first time after that change, it showed an alert pop-up saying that something went wrong and the configurations were to be restored. After that, we did not notice any difference when working with Chromium while surfing the web, installing/uninstalling browser extensions, adding plugins, modifying the homepage, adding bookmarks or adding/deleting pinned tabs.

However, even if we generate random seeds to mitigate the attack, the problem remains if the nonce is still stored in the file system. We thus implement a script to generate the random seed each time Chromium is about to open. The problem with this solution is that Chromium became impractical since it was always trying to restore the file from external changes (remember that the Secure Preferences file is analyzed and loaded each time Chromium is launched). If the seed is changed, the HMAC protection mechanism implemented by the Secure Preferences file should also be updated, generating thus new values for all the

`macs` as well as for the `super_mac`. The conclusion is that the seed generation is not secure as long as it is stored somewhere in the file system of the user: performing a reverse engineering process is enough to reveal where the seed is stored, being therefore easy to get it.

Briefly, either the seed generation, the Secure Preferences file or the seed storage have to be protected from unauthorized parties. To achieve that we propose a solution based on (TPM), in which case the seed should automatically be generated and stored in a secure memory so only the browser can access it (as Windows 10 currently does [28]). A limitation of the usage of TPMs, despite being widely extended, is that not all computers have one.

Alternative solutions to TPM, e.g., Intel SGX, ARM Trusted Zone or MAC secure enclave, could be considered. In such cases, the browser can either partially or totally run the seed generation procedure in the enclave and securely store the generated seed. Moreover, the browser could also store the Secure Preferences file in the enclave so no other parties different than the browser can access it.

Potential Benefits. Creating a controlled environment to execute browser extensions and analyze them is difficult. Honey pages have been widely used in the literature to fire the execution of browser extensions [9, 19, 36]. Our released source code can easily be used to modify extensions in such a way that the main functionality is not modified. Therefore, applying techniques like fuzzing, improving static analysis strategies or making dynamic analysis less demanding are some of the examples where our changeware can be helpful.

5 Related Work

Many researchers have analyzed browser extensions from the security and privacy point of view (e.g., [5, 8, 13, 16, 19, 24, 32, 35, 36, 44]) but very little research has been conducted about how browser preferences and the Secure Preferences file can be used by malicious software to attack user's privacy or security.

The first attack against the Secure Preferences file, on Chrome for Windows, was described in one Internet forum in 2015, where it was shown how this file could be silently modified [17]. We confirmed this and developed a new attack based on that one that combined can be used to modify the Secure Preferences file of any version of any Chromium-based browser. Indeed, we turned a less known narrowly-targeted attack (that only worked for Chrome and only on Windows) into a powerful platform-independent attack that exploits the most important Chromium-based browsers. Furthermore, we presented a systematic study of this class of attack and investigated its hefty consequences for browser hijacking and browser extensions.

In the same year, Banescu *et al.* [6] assumed the existence of a type of malware called changeware with no root privileges. This malware is typically installed by Internet toolbars, banners or the execution of executable files like installers whose goal is to change user's configuration files. In this paper, Banescu *et al.* proposed a solution based on White-Box Cryptography. As this type of cryptographic tool is insecure [7, 33], they include software diversity [14] to mitigate attacks against this cryptographic scheme [6]. Since the attackers are not aware of the used obfuscation transformation, they need to explore all the possible generated binaries to run cryptanalysis. Despite being a promising technique, the proposed solution is only deployable in Windows since they do not modify the kernel of the operating system. A modification of the kernel would be mandatory in Linux and Mac. In comparison to our paper, we focused on how to perform the attack against the Secure Preferences file and the consequences for the user.

In most, if not all, the referenced papers try to find security solutions for browser extensions without being concerned about the entry point of these preferences in the browser. Active extensions, web accessible resources, permissions they have, silent (un)installations or the path of installation where all the files and extra files are located in the OS are a few examples of topics covered in the literature. We went one step forward and described an attack to the Secure Preferences file where all the preferences of the user are stored. We can actually modify any of these settings and thus bypassing most of the proposed solutions in the literature, originating new security and privacy issues.

6 Conclusions

We have revisited the security and privacy of Chromium's mechanism to access the Secure Preferences file. Google introduced a security mechanism based on a cryptographic algorithm named HMAC to avoid users and applications other than the browser modifying the Secure Preferences file. We found that the seed used for the HMAC is fixed, making Chromium vulnerable to PUP. Our analysis was carried out on Brave, Chrome, Edge and Opera.

We have also demonstrated that it is possible to perform browser hijacking, browser extension fingerprinting and remote code execution attacks as well as silent browser extensions (un)installation. We did so by coding a platform-independent proof-of-concept changeware that exploits the HMAC, freely modifying the Secure Preferences file. Our changeware, in combination with the one proposed in [17], can be used to modify such a preferences file of any Chromium version later than v.25 (including the latest one, v.85.0).

Acknowledgments. This work was partially supported by the Swedish Foundation for Strategic Research (SSF) and the Swedish Research Council (Vetenskapsrådet) under grant Nr. 2015-04154 (PoUser: Rich User-Controlled Privacy Policies).

A Installed-by-Default Extensions

Table 3. Brave installed-by-default extensions.

ExtensionID	Name	Uninstallable
ahfgeienlihckogmohjhadljkjgocpleb	Web Store	✗
jidkidbbcafbjadbphckchenhfomhmfma	Brave Rewards	✗
kmendfapggjehodndflmmgagdbamhdfd	CryptoTokenExtension	✗
lgjmpdmojkpcjocpdikifhejkkjglho	Brave Webtorrent	can be disabled
mfehgcgbbipciiphmccgaenjdiccnmng	Cloud Print	✗
mhjfbmdgcfjbbpaeojofohoefgiehjai	Chromium PDF Viewer	✗
mnojpmjdmdbbfmejpfflffihffcmidifd	Brave	✗
nkeimhogjdpnpccoofpliimaahmaaome	Google Hangouts	can be disabled
odbfpeeihdkbihmopkbjmoonfanlbfcl	Crypto Wallets	can be disabled
oemmndcblldboiebnladdacbfmadadm	PDF Viewer	✗

Table 4. Microsoft Edge installed-by-default extensions.

ExtensionID	Name	Uninstallable
dgiklkfkllickanfonkcabmbdfmgleag	Edge Clipboard	✗
fikbjbembnmfhpjfnmfkahdhfohhjmg	Media Internals Services Extension	✗
fogppepbmgkpdkinbojbibkhoffpief	Edge Collections	✗
iglcjdemknebjbklcgkfaebgojjphkec	Microsoft Store	✗
ihmafllikibpmigkcoadcmckbbfhibefp	Edge Feedback	✗
jdicldimpdaibmpdkjnbmckianbfold	Microsoft Voices	✗
kmendfapggjehodndflmmgagdbamhdfd	CryptoToken	✗
mhjfbmdgcfjbbpaeojofohoefgiehjai	Microsoft Edge PDF Viewer	✗
ncbjelpjchkbpbkckchkhkblodoama	WebRTC Internals Extension	✗
nkeimhogjdpnpccoofpliimaahmaaome	Google Hangouts	✗
pkedcjkdefgpdelpcbmbmeomcjbbeemfm	Chrome Media Router	✗

Table 5. Chrome installed-by-default extensions.

ExensionID	Name	Uninstallable
aapocclcgogkmnckokdopfmhfonfngoek	Slides	✓
ahfgeienlihkogmohjhadlkjgocpleb	Web Store	✗
aohghmighlieiainnegkcijnfilokake	Docs	✓
apdfllckaahabafndbheiahigklhal	Google Drive	✓
blpcfgekakmgkcojhhkbfldkacnbeo	Youtube	✓
felcaaldnbdnccldmndcnolpebgiejap	Sheets	✓
gfdkimpbcpaahaombhbimeihdjnejgicl	Feedback	✗
ghbmnnjooekpmoecnnlnnbdlolhkhki	Google Docs Offline	✓
kmendfapggehodnddfmmgagdbamhnfd	CryptoTokenExtension	✗
mfehgcgbbipicphmccgaenjdiccnmng	Cloud Print	✗
mhjfbmdgcfjbbpaeojofohoefgihjai	Chrome PDF Viewer	✗
neajdpdkcdipfabeoofebfddakdcjhd	Google Network Speech	✗
nkeimhogjdpnpccoofpliimaahmaaome	Google Hangouts	✗
nmmhkkegccagdldgiimedpiccmgmieda	Google Wallet	✗
pjkljhegncpnkpbcohdijeoejaedia	Gmail	✓
pkedcjkdefgpdelpbcmbmeomcjbeemfm	Chrome Media Router	✗

Table 6. Opera installed-by-default extensions.

ExensionID	Name	Uninstallable
apkgpnbdlipaagpckkbdbigfmmomobn	Onboarding popup	✗
bcibcaakpeekhbnddgajbjmjdcmfkl	Opera Addons Portal	can be disabled
bennllbledkboeijomefbhpidmhfkoih	News feeds popup	✗
cgloicgndbkhmjcaddholfcgghcgmimg	Opera Welcome Page	✗
eiccfcfdclpgnaagpkjfpkaabgcbne	SD suggestions list	✗
efpeldimhbhjejgcdcbhmjllaaahjmge	Vkontakte Notifications	✗
enmlgamfkfdemjmlfjeieipglcfpomikn	News feed handler	✗
gfobfmjpcnapngbghpcbodncehngmdl	Opera Crypto Wallet	✗
hhckidpbkbmoejbdobdgdialionif	Video handler	✗
ibgcfekaajeggoajjnmknjcoieffdnod	Google Drive/Docs clipboard and notifications support	✗
ionkhgehfolinkdpgdbinmgbfaonpcnk	Amazon promotion	✗
jaocpocicpmhbcchlodkiochdkmophj	Aliexpress observer	✗
kmendfapggehodnddfmmgagdbamhnfd	CryptoTokenExtension	✗
knohfehbibecknbfoecpdmkdjkjdjnl	Bookmarks	✗
mfglbjdihkhhnimleciocbjbiepicip	Opera Sync Auth Flow	✗
mhjfbmdgcfjbbpaeojofohoefgihjai	Chromium PDF Viewer	✗
midfadfpkkakgcbgpnfgnfekghlige	Rate Opera	✗
nkeimhogjdpnpccoofpliimaahmaaome	Google Hangouts	✗
obhaigpnhcioanniiaepcgkdilopflbb	Background worker	✗
odndjknigpndmldfodecoelobjidna	Opera In-App Notification Portal	✗
onigllbobbpnlncjanphobocbkcdghh	Discord Notifications	✗

References

1. spyware: Softonic (2019). <https://www.2-spyware.com/remove-softonic.html>
2. Aggarwal, A., Viswanath, B., Zhang, L., Kumar, S., Shah, A., Kumaraguru, P.: I spy with my little eye: analysis and detection of spying browser extensions. In: EuroS&P, pp. 47–61, April 2018
3. Arshad, S., Kharraz, A., Robertson, W.: Identifying extension-based ad injection via fine-grained web content provenance. In: Monrose, F., Dacier, M., Blanc, G., Garcia-Alfaro, J. (eds.) RAID 2016. LNCS, vol. 9854, pp. 415–436. Springer, Cham (2016). <https://doi.org/10.1007/978-3-319-45719-2.19>
4. Awakesecurity: Discovery of a massive, criminal surveillance campaign (2020). <https://awakesecurity.com/blog/the-internets-new-arms-dealers-malicious-domain-registrars/>
5. Bandhakavi, S., Tiku, N., Pittman, W., King, S.T., Madhusudan, P., Winslett, M.: Vetting browser extensions for security vulnerabilities with VEX. *Commun. ACM* **54**(9), 91–99 (2011)
6. Banescu, S., Pretschner, A., Battre, D., Cazzulani, S., Shield, R., Thompson, G.: Software-based protection against changeware. In: CODASPY, pp. 231–242 (2015)
7. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential computation analysis: hiding your white-box designs is not enough. In: CHES, pp. 215–236 (2016)
8. Carlini, N., Felt, A.P., Wagner, D.: An evaluation of the google chrome extension security architecture. In: USENIX, pp. 97–111 (2012)
9. Chen, Q., Kapravelos, A.: Mystique: uncovering information leakage from browser extensions. In: CCS, p. 1687–1700 (2018)
10. Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.: White-box cryptography and an AES implementation. In: Selected Areas in Cryptography, pp. 250–270 (2003)
11. Chromium: No more silent extension installs (2019). <http://blog.chromium.org>
12. Cimpanu, C.: Windows 10 to get PUA/PUP protection feature (2020). <https://www.zdnet.com/article/windows-10-to-get-puapup-protection-feature/>
13. Dhawan, M., Ganapathy, V.: Analyzing information flow in Javascript-based browser extensions. In: ACSAC, pp. 382–391 (2009)
14. Forrest, S., Somayaji, A., Ackley, D.H.: Building diverse computer systems. In: Workshop on Hot Topics in Operating Systems, pp. 67–72, May 1997
15. gs.statcounter: Browser market share (2020). <https://gs.statcounter.com/browser-market-share>
16. Guha, A., Fredrikson, M., Livshits, B., Swamy, N.: Verified security for browser extensions. In: S&P, pp. 115–130 (2011)
17. HMAC: Chromium Secure Preferences (2019). <https://kaimi.io/2015/04/google-chrome-and-secure-preferences/>
18. Jagpal, N., et al.: Trends and lessons from three years fighting malicious extensions. In: USENIX, pp. 579–593 (2015)
19. Kapravelos, A., Grier, C., Chachra, N., Kruegel, C., Vigna, G., Paxson, V.: Hulk: eliciting malicious behavior in browser extensions. In: USENIX, pp. 641–654 (2014)
20. Karami, S., Ilija, P., Solomos, K., Polakis, J.: Carnus: exploring the privacy threats of browser extension fingerprinting. In: NDSS (2020)
21. Kotzias, P., Matic, S., Rivera, R., Caballero, J.: Certified pup: abuse in authentic-code code signing. In: CCS, pp. 465–478 (2015)
22. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: keyed-hashing for message authentication. Internet Engineering Task Force (IETF) (1997)

23. Laperdrix, P., Bielova, N., Baudry, B., Avoine, G.: Browser fingerprinting: a survey. CoRR abs/1905.01051 (2019). <http://arxiv.org/abs/1905.01051>
24. Lerner, B.S., Elberty, L., Poole, N., Krishnamurthi, S.: Verifying web browser extensions' compliance with private-browsing mode. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 57–74. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40203-6_4
25. Malwarebytes: Billion-dollar search engine industry attracts vultures, shady advertisers, and cybercriminals (2020). <https://blog.malwarebytes.com>
26. Malwarebytes: WinYahoo (2020). <https://blog.malwarebytes.com>
27. Microsoft: Microsoft edge: making the web better through more open source collaboration (2019). <https://bit.ly/2QeZFwm>
28. Microsoft: How windows 10 uses the trusted platform module (2020)
29. Microsoft: Windows defender and secure preferences file (2020). <https://answers.microsoft.com>
30. Picazo-Sanchez, P., Tapiador, J., Schneider, G.: After you, please: browser extensions order attacks and countermeasures. *Int. J. Inf. Securi.* 1–16 (2019)
31. Rogowski, R., Morton, M., Li, F., Monrose, F., Snow, K.Z., Polychronakis, M.: Revisiting browser security in the modern era: new data-only attacks and defenses. In: EuroS&P, pp. 366–381, April 2017
32. Sánchez-Rola, I., Santos, I., Balzarotti, D.: Extension breakdown: security analysis of browsers extension resources control policies. In: USENIX, pp. 679–694 (2017)
33. Sanfelix, E., Mune, C., de Haas, J.: Unboxing the white-box. In: Black Hat EU 2015 (2015)
34. Sjösten, A., Van Acker, S., Picazo-Sanchez, P., Sabelfeld, A.: LATEX GLOVES: protecting browser extensions from probing and revelation attacks. In: NDSS (2018)
35. Somé, D.F.: Empoweb: empowering web applications with browser extensions. In: S&P, pp. 227–245, May 2019
36. Starov, O., Nikiforakis, N.: Xhound: quantifying the fingerprintability of browser extensions. In: S&P, pp. 941–956 (2017)
37. Starov, O., Laperdrix, P., Kapravelos, A., Nikiforakis, N.: Unnecessarily identifiable: quantifying the fingerprintability of browser extensions due to bloat. In: WWW, p. 3244–3250 (2019)
38. Statcounter: Desktop Browser Market Share Worldwide (2019). <https://gs.statcounter.com>
39. UK, P.: Update Java, get yahoo as your default search engine (2019). <https://uk.pcmag.com>
40. Urban, T., Tatang, D., Holz, T., Pohlmann, N.: Towards understanding privacy implications of adware and potentially unwanted programs. In: ESORICS, pp. 449–469 (2018)
41. Varshney, G., Misra, M., Atrey, P.K.: Detecting spying and fraud browser extensions: short paper. In: MPS, pp. 45–52 (2017)
42. w3schools: Browser Statistics (2019). <https://www.w3schools.com/browsers/>
43. Xing, X., et al.: Understanding malvertising through ad-injecting browser extensions. In: WWW, pp. 1286–1295 (2015)
44. Zhao, R., Yue, C., Yi, Q.: Automatic detection of information leakage vulnerabilities in browser extensions. In: WWW, pp. 1384–1394 (2015)



Detecting Word Based DGA Domains Using Ensemble Models

P. V. Sai Charan^(✉), Sandeep K. Shukla, and P. Mohan Anand

Department of Computer Science and Engineering, Indian Institute of Technology,
Kanpur, Kanpur, India
{pvcharan,sandeeps,pmohan}@cse.iitk.ac.in

Abstract. Domain Generation Algorithm (DGA) is a popular technique used by many malware developers in recent times. Nowadays, DGA is an evasive technique used by many of the Advanced Persistent Threat (APT) groups and Botnets to bypass host and network-level detection mechanisms. Legacy malware developers used to hard code the IP address of control and command server in malware payload. But, this led to identifying malicious IP address by reverse engineering the malware payload. Drawbacks in this hardcoding IP mechanism led to the idea of character-based Domain Generation Algorithms, where attackers generate a list of domain names using traditional cryptographic principles of pseudo-random number generators (PRNGs). Recent advances in malware research, machine learning address this problem to a large extent. Lately, malware developers came up with a new variant of DGA called word-list based DGA. In this approach, the malware uses a set of words from the dictionary to construct meaningful substrings that resembles real domain names. In this paper, we propose a new method for detecting Word-list based DGA domain names using ensemble approaches with 15 features (both lexical and network-level). Added to this, we generated syntactic data using CTGAN (GAN-based data synthesizer that can generate synthetic data) to measure the robustness of our model. In our experiment, C5.0 stands out as the best with prediction accuracy of 0.9503 and out of 30000 synthetically generated malicious domains names, 1351 classified as benign.

Keywords: DGA · APT · Malware · GAN · Diffusion map · PCA

1 Introduction

Modern-day malware are intelligent enough to hide their presence by applying various advanced evasive techniques [1]. One such popular technique is Domain Generation Algorithms. As the process of embedding this technique into malware code is easy and effective, we can notice the implementation of the same in many advanced malwares such as APT's and Ransomware [2]. The important aspect of any malware is to stay undetected for the longest possible period. This part can be accomplished at various levels of malware implementations.

Obfuscating code is the most common technique at the code level. But in the communication part, most of the malware families are vulnerable to detection at the firewall level as it tries to communicate to a static IP address or a domain name throughout its lifetime. It's very obvious that hitting the same IP/domain name frequently by a process can be treated as abnormal behavior and a single line of firewall rule can be effective in blocking such network traffic. So, for less conspicuous communication between a malware and its C&C (control and command server); DGA (Domain Generation Algorithms) came into existence [3]. Domain Generation Algorithms can be treated as an enhancement functionality written inside the malware piece of code which generates a dynamic list of domain names that changes over time. In other words, a DGA present inside a malware generates a list of domain names which the attacker registers with DNS registry for a fixed period. This way of generating domain names makes the job difficult for the firewall to detect suspicious communication.

In the initial step, attacker tries to inject the malware into a vulnerable host system via Phishing, Spam mail content, host based code injection etc [4]. After getting settled inside the host system, malware starts communicating with its C&C server. At this point, DGA comes into the picture. Typically, the DGA work-flow starts from the attacker's side by running the algorithm with a seed value (timestamp, currency exchange rate between two countries, trending hashtag in twitter, etc) and generates a list of domain names. As the attacker knows that the same set is generated by the malware at the end-user level, attacker registers a few of those domain names and use them as C&C servers. In this process important aspect lies in activating the DGA domain names for a short period with a constant seed value change.

Character-Based DGA is one of the basic implementations where pseudorandom strings are generated and a legitimate top level domain (TLD) is concatenated to make it look like an actual domain name (eg: wxfuyhpdfkzsh.com) [5]. Here, the process of generating the pseudorandom string is dependent on the seed value. Kraken was the first infamous malware to be found at the beginning of 2008 [6]. Later, a malware named Conficker affected more than 11 million devices world wide by installing scareware content in windows machines [7]. In 2009, another DGA malware named Zeus had impacted 70,000 bank and business accounts including the NASA [8]. Pykspa was another significant DGA malware that used Skype to damage the victim's computer [9]. Most of these character-based DGA domain name families are a bit easy to detect as their output domain names look quite different from the normal ones. There are various attributes like n-gram statistics, vowel-consonant pairing, unique words etc helps in classifying these character based DGA domain names with a decent accuracy [10].

More recently, the malware writers moved to a word-list based DGA, where two or three words from different word-lists are selected and concatenated in random. Finally, a TLD is added at the end that resembles a normal domain name (eg: crossmentioncare.com) [11]. Rovnix is one such legacy DGA family which generates command and control (C&C) domains using words from the United

States Declaration of Independence, the GNU Lesser General Public License and other documents [12]. According to security researchers, evolutionary DGA families like Matsnu employs a clever technique to evade conventional detection mechanisms [13]. Matsnu DGA generates 16–24 character domain names based on a blend of names and verbs (noun-verb-noun-verb). The words used by the malware can be entered by the attacker or taken from a predefined list containing 878 nouns and 444 verbs. This Matsnu DGA is configurable, so that it allows cyber criminals to set the number of domains they want to create regularly [13]. In this, the DGA malware writer is given a choice to specify the number of days after which the previously generated domain name is ready for re-use. The Matsnu DGA family has been actively used by several new malware variants since June 2014. The highest number of infections has been detected in Germany (89%), however, some of the infected devices are located in Austria and Poland as well [14]. Recently, one Security firm has sinkholed one of the servers used by Matsnu and found roughly 9,000 bots communicate with it each day.

Although many traditional and hybrid methods are available in the detection of character-based DGA, usage of Word list based DGA in malware attacks is rapidly increasing globally. Attackers created evolutionary word list based DGA families like Matsnu, Gozi and suppobox which are resistant to traditional detection techniques because of their proximity to real world domain names. To address this serious security concern, we propose a detection method for word list based DGA domain names (Matsnu, Gozi and Suppobox) using the Ensemble Learning Algorithms and evaluate the efficiency of these algorithms. In our experiments, we applied both linear and non-linear dimensionality reduction techniques to analyse underlying structure of our data. In addition, we generated synthetic tabular data using CTGAN to measure the robustness of our model.

This paper is organized as follows. Section 1 details about overview of DGA and its types followed by the challenges that corporate networks encounter with new generation DGA domain names. Section 2 explains Related work and problems involved in current methodologies in detecting word list based DGA. Section 3 briefs on Proposed approach for extracting features and building classifiers to detect DGA domain names. Section 4 demonstrates Implementation and results for various experiments. Section 5 details Conclusion and future work.

2 Related Work

As Domain Generation Algorithms became a common evasion technique for attackers in recent years, DGA detection became significantly important in modern corporate networks. Traditional Domain Generation Algorithms uses pseudo-random number generators to generate a list of domains. But this area of research is widely explored by many researchers and got benchmarking results in detecting PRNG based DGA domain names. Yadav proposed a method to detect DGA domain names using temporal correlation. They have mainly considered the IP-domain bipartite graph and information entropy of bigrams [15]. Similarly, da

Luz et al. built a model from passive DNS data by considering lexical and network features [16]. In 2019, Jose proposed a masked n-gram model in which they considered n-grams for second-level domain name values [17]. Recent advances in machine learning algorithms improved detection rates of PRNG based DGA domain names. Especially, ensemble models like C5.0 and Random Forests performs the best in detecting these with very high accuracy and less false-positive rates.

On the other hand, word-based Domain Generation Algorithms detection seems challenging to most of the security experts and researchers. In 2016, Daniel Plohmann et al. did a comprehensive survey on Domain Generative Malware [18]. Their work clearly explains the complexity of word-list based domain generation algorithm families like Matsnu, Suppobox and Gozi. Curtin et al. proposed a model for detecting DGA domain names with recurrent neural networks [19]. In their model, they used a concept called smashword score (which measures how much a DGA family is close to the English words) to train recurrent neural network and generalized likelihood ratio test (GLRT) LSTM to detect malicious word list based domain names. But their accuracy is not up to the mark for Matsnu, Suppobox and Gozi families. Because of this reason, it is not adaptable for a large scale sensitive corporate networks.

Similarly, Luhui Yahg et al. proposed a method for detecting word-list based DGA by considering word feature, part-of-speech feature and word correlation feature [20]. Their work mainly concentrated on Front-Word-Correlation (FWC) and Back-Word-Correlation (BWC) in the word correlation analysis. Although they used ensemble models like J48 from weka3.8, their accuracy (0.83) remains poor with high false positives. So, this model is not feasible for current industrial standards. In 2016, Woodbridge et al. proposed a model to predict Domain Generation Algorithms using LSTM neural networks [21]. Although their approach needs no manual feature extraction and less classification time, their model have class imbalance that limits its ability to detect families like Suppobox and Matsu.

Choi et al. proposed the BotGAD framework which captures all the DNS traffic passing at the network level and creates a matrix of timeslots and IP address requested [22]. BotGAD main focus is on Time To Live of DNS record which is not self-sufficient to detect sophisticated Botnet or APT evasion techniques. In a similar way, Jasper has proposed the DGA detection method based on the popularity of the domain name [23]. In this approach, a sudden increase and decrease of traffic flow to a particular domain is monitored over a period of time and based on that, the popularity of a domain name is calculated. This approach will take a minimum of 1 day to observe the changes in a network pattern which is infeasible for real-time detection. Luhui et al. proposed a model to detect word-based DGA using semantic analysis [20]. In this work, they mainly considered word embeddings of parts of speech, inter-word correlation, and inter-domain correlation for constructing a machine learning model. Although their work analyzed frequency distributions of words and parts of speech of Domains names, the major drawback with this model is a large number of features that

are interdependent of each other and not even a single DNS based feature is considered. These are the reasons behind the accuracy drop for this model even after using ensemble classifiers.

3 Proposed Methodology

We collect our word list based DGA samples from DGArchive (database for domain names that are dynamically created by malware using Domain Generation Algorithms). Similarly, we take data of legitimate domain names from Alexa one million domain names list dataset. In our experiments, we mainly concentrated on Gozi, Matsnu and Suppobox DGA families (which are difficult to detect by many state of the art DGA detection mechanisms). We extract 15 features (lexical and network features) for both legitimate and malicious domain names (specified in Table 1). In this feature extraction process, we have used the wordninja library for splitting words to extract lexical features [24]. Wordninja is based on probabilistically split concatenated words using NLP on English Wikipedia unigram frequencies. Similarly, we used the whois (importable Python module which will produce parsed WHOIS data for a given domain) library for extracting network-level features [25]. This library helps us to extract data for popular TLD's by sending a direct query to a WHOIS server instead of going through an intermediate web service.

3.1 Building Classifiers

After preparing a dataset of 15 features for randomly selected domain names from the Matsnu, Gozi, Suppobox and Alexa domains, the next phase is to build a low false positive classifier. Initially, we build Naive Bayesian. But this Naive Bayesian model misclassified most of Gozi and Suppobox families as legitimate domain names which resulted in poor accuracy and high false positive rate. Later, we build KNN classifier (distance functions based approach) to detect word-list DGA families. KNN model performed well in classifying DGA domain names, but this classifier raised false alarms by misclassifying legitimate domains as DGA families. As the process of word-list DGA domains detection requires near real time detection with less misclassification errors, we use ensemble approach to construct classifiers. The idea of an ensemble approach is to build a strong learner by combining several weak learners. Basically, ensemble approaches are classified as Boosting, Bagging and Stacking respectively. Boosting works in an iterative way where more weights are assigned to the misclassified learners, and in the next iteration errors are minimised. We use GBM (stochastic gradient boosting), J48 (C4.5), C5.0 as boosting models and CART, Random Forest as bagging models. Boosting ensemble construct trees sequentially to minimize the errors of the previous trees. Unlike boosting ensemble models, in bagging ensemble models trees are built in parallel and final model is built by aggregating predictions from all models. So, Boosting ensemble models gave better results in detecting word-list based DGA domain detection. In order to achieve a higher

accuracy value, C5.0 uses Information gain for the selection of attributes to construct a decision tree and Binomial Confidence Limit as a pruning technique. Similarly, C4.5 generates decision trees from a collection of training data like ID3, using the knowledge entropy principle. In C4.5, all errors are regarded as equivalent, although some classification errors are more severe than others in practical applications. C5.0 requires a different cost to be defined for each predicted/actual class pair; C5.0 constructs the classifiers to reduce the expected misclassification costs rather than the error rate. On the other hand in GBM, new basic learners are created which can be correlated with the overall negative gradient of the loss function.

Table 1. Features considered for sample MATSNU domain name

S.NO	Feature	Example (crossmentioncare.com)
1	Domain Name	crossmentioncare.com
2	Word Count	3
3	Length	16
4	Syllable Count	4
5	Vowel Count	6
6	Consonant Count	10
7	Created Since (in days)	2192
8	Updated Since (in days)	2189
9	Registrar (Binary)	1
10	TTL (in seconds)	86400
11	IANA (Binary)	1
12	Unique Letters	10
13	Hyphen (Binary)	0
14	Underscore (Binary)	0
15	Family Type	MATSNU

We use CARET package in R programming language to build models in all our experiments except Naive Bayes. Since we have both categorical and continuous data in our dataset, for building a Naive Bayes classifier we use Mixed Naive Bayes python library [26]. We use repeated cross-validation with 10 resampling iterations as the train control method and this step is repeated 3 times for each classifier. As Figs. 1 and 2 show, our model provides substantial accuracy improvements as compared to previous approaches in detecting word-list based DGA domain names. In Fig. 1. (Accuracy comparison graph), Accuracy values taken along X-axis and various classifiers are taken along Y-axis. Similarly in Fig. 2 (Kappa comparison graph), Kappa values taken along X-axis and various classifiers are taken along Y-axis respectively. Building blocks for our model are

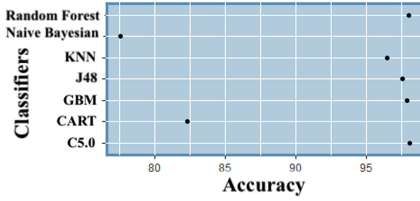


Fig. 1. Accuracy comparison graph

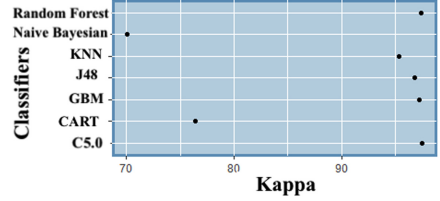


Fig. 2. Kappa comparison graph

the features listed in Table 1. Figure 3 displays the Receiver Operating Characteristic (ROC) curves generated by various classifiers. ROC graph depicts the relation between True Positive Rate and the False Positive Rate. All the ensemble classifiers have a fairly equal Area under Curve (AUC) value and for the C5.0 classifier, the maximum accuracy of 0.9808 is achieved.

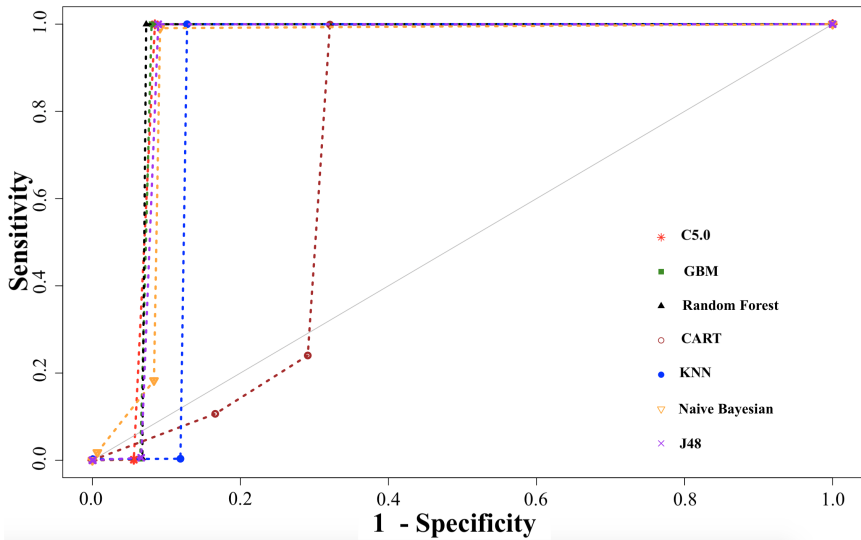


Fig. 3. ROC curve for various Classifiers

4 Results and Discussion

After data engineering and basic model construction part, we carry out experiments to reduce the feature set and improve the accuracy. In our paper, we implement feature correlation analysis to identify the highly correlated feature set which helps in the feature reduction part. In addition, we implemented both linear and nonlinear dimensionality reduction methods like PCA (Principal Component Analysis) for linear feature reduction and Diffusion map to discovering

the underlying manifold of the dataset [27, 28]. The following subsections describe these experiments.

4.1 Experiment-1

Initially, we consider all 15 features for model training. In this experiment, we consider 40,000 domain samples (10,000 random samples from each type i.e Matsnu, Gozi, Suppobox, Benign). These 40,000 sample data set is divided into 60:40 for training and testing phases. Then we applied various ensemble models on this dataset and according to our observation, most of the ensemble models are well suited for near real-time predictions for these word list based DGA domains prediction. Among these ensemble models, C5.0 stands out to be the best one with low training time and high accuracy. In Table 2, Benign(B), Matsnu(M), Gozi(G), Suppobox(S) are respective DGA family types in sensitivity and Specificity columns. Kappa statistic is a measure of how closely the instances classified by the machine learning classifier match the data labeled as ground truth and Sensitivity is a metric that evaluates the ability of the model to predict the true positive of each DGA family category. Similarly, Specificity deals with true negative values of each DGA family. In our observation, C5.0 is best among all other classifiers with low False Positive Rate (FPR) and a low False Negative Rate (FNR) for 15 features dataset.

Table 2. Results obtained after applying various classifier models on 15 feature dataset

Algorithm	Accuracy	Kappa	Sensitivity	Specificity
Naive Bayesian	0.7758	0.7011	0.9130(B), 0.9960(M), 0.8060(G), 0.3975(S)	0.9742(B), 0.7717(M), 0.9646(G), 0.9914(S)
KNN	0.9650	0.9532	0.8718(B), 0.9960(M), 0.9965(G), 0.9968(S)	0.9988(B), 0.9591(M), 0.9974(G), 0.9972(S)
CART	0.8230	0.7640	0.8790(B), 0.9200(M), 0.7580(G), 0.9300(S)	0.9997(B), 0.9138(M), 0.9426(G), 0.9131(S)
GBM	0.9789	0.9719	0.9197(B), 0.9998(M), 0.9960(G), 1.00(S)	1.00(B), 0.9780(M), 0.9953(G), 1.00(S)
Random Forest	0.9802	0.9737	0.9277(B), 0.998(M), 0.9950(G), 1.00(S)	0.9999(B), 0.9758(M), 0.9977(G), 1.00(S)
C 5.0	0.9808	0.9744	0.9745(B), 1.00(M), 0.9980(G), 1.00(S)	1.00(B), 0.9808(M), 0.9943(G), 1.00(S)
J48	0.9756	0.9675	0.9110(B), 0.997(M), 0.993(G), 1.00(S)	0.9998(B), 0.9763(M), 0.9920(G), 1.00(S)

When we do a feature correlation analysis by constructing a feature correlation plot for our 15 feature dataset, (inspired by Tian Zheng, Matthew Salganik and Andrew Gelman’s work on estimation of social structure in the network by using overdispersion count [29]) we get a correlation plot as shown in Fig. 4. We understand how one feature in our dataset is correlated to all other features with an index ranging from -1 to 1 . $+1$ indicates a strong correlation (colored as dark red) and -1 indicates a strong negative correlation (colored as dark blue).

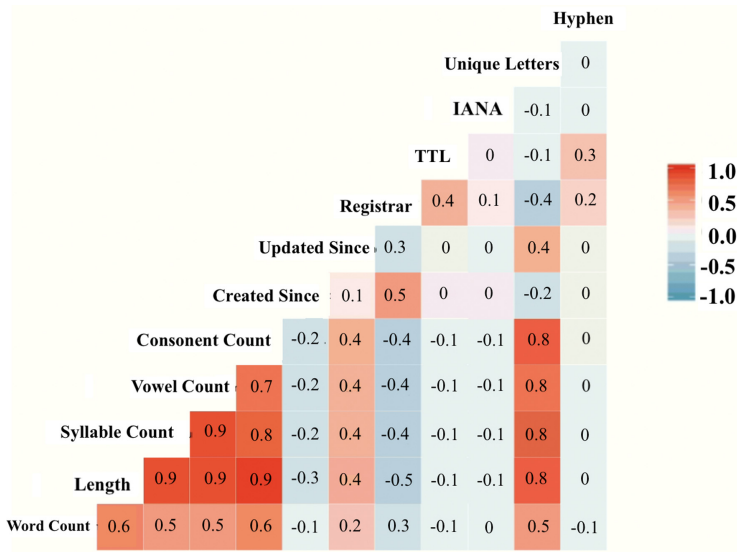


Fig. 4. Feature correlation analysis for 15 features dataset (Color figure online)

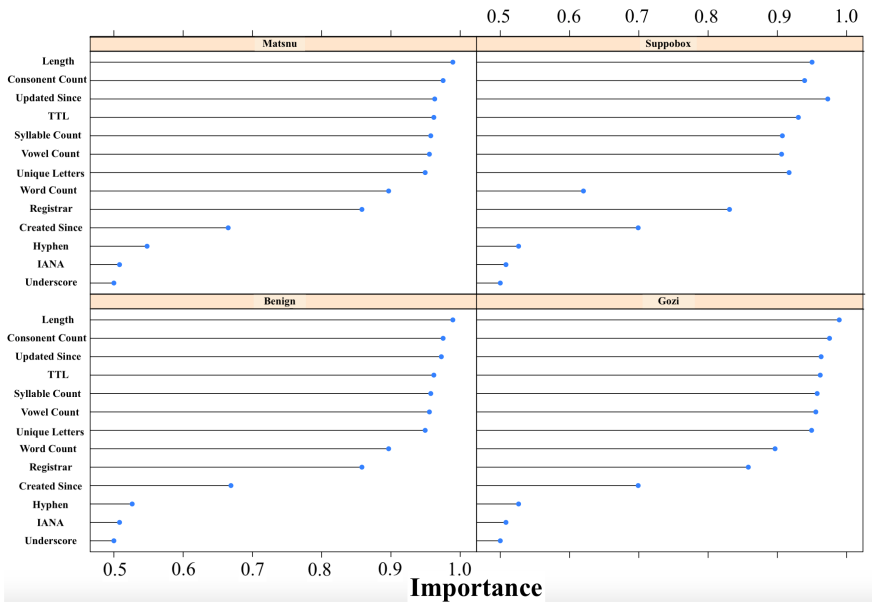


Fig. 5. Feature importance graph for 15 features

Along with feature correlation analysis, we construct a feature importance graph for our dataset to understand the significant features. This feature correlation plot is constructed based on recursive feature elimination method [30]. In this method, variable importance is computed using the ranking method for feature selection. For the final subset of features, importance across all resamples are averaged to compute an overall importance value for the features. For our dataset of 15 features, the feature importance graph is illustrated in Fig. 5. In this feature importance graph, feature importance value (lies between 0 and 1) is taken along X-axis and Feature list is taken along Y-axis respectively.

4.2 Experiment-2

Based on these two results, we consider top 8 features (4 lexical and 4 network based) and train a new model. For this experiment, we also consider 40,000 domain samples (10,000 random samples from each type i.e Matsnu, Gozi, Suppobox, Benign) divided in 60:40 ratio for training and testing respectively. We apply various ensemble models on the data, Random forest tops in terms of high accuracy and low FPR, FNR rates. We achieve almost similar accuracy (2% drop) by reducing half of the features as shown in Table 3. Although Random forest tops in terms of accuracy, its training time and model size is double than the C5.0. If we are looking for real time DGA domain name detection for a sensitive corporate network’s that needs a daily or weekly retraining, C5.0 is the best choice with less model size and training time as well. After constructing models with 8 features we did a similar feature importance analysis for this 8 features as illustrated in Fig. 6. In this fea-

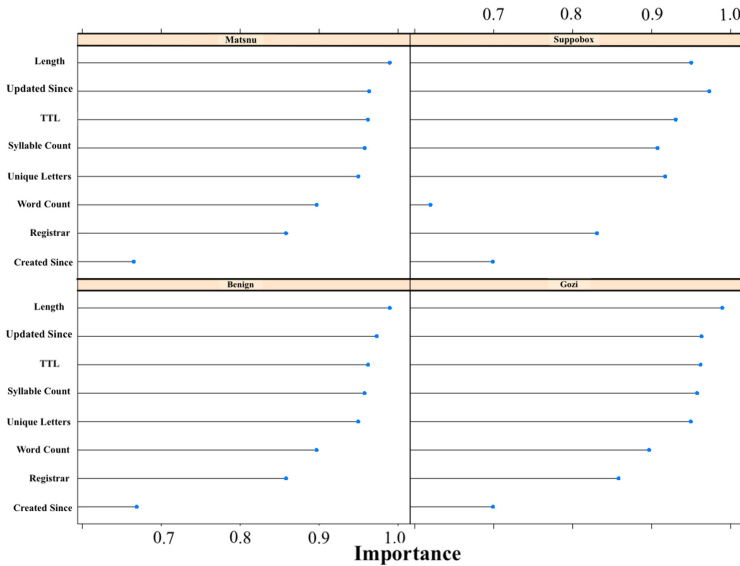


Fig. 6. Feature importance graph for 8 features

ture importance graph (8 features), feature importance value (lies between 0 and 1) is taken along X-axis and Feature list is taken along Y-axis respectively.

4.3 Experiment-3

In this experiment, we apply one of the linear dimensionality reduction technique called as Principle Component Analysis (PCA) on our 15 feature dataset. Basic idea of PCA is to maximize the variance of first K components and minimize the variance of remaining P-K components by projecting the data from P-dimensional space to K-dimensional sub space. Initially, we plot a graph between Principle Components (along X-axis) and cumulative sum of variance (along Y-axis) to understand top K significant principle components as shown in Fig. 7.

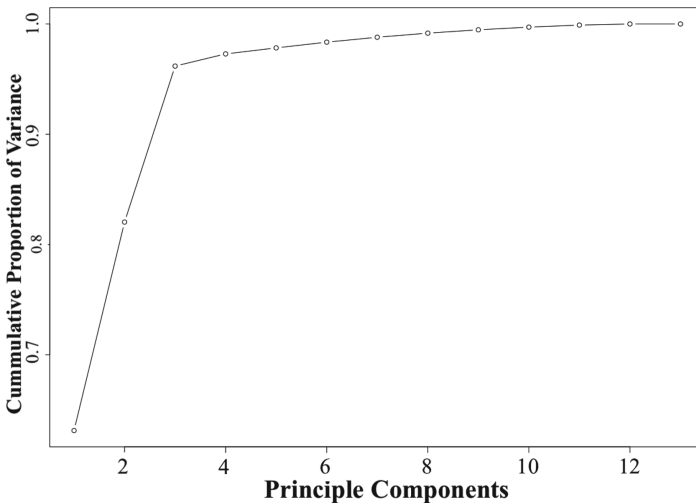


Fig. 7. Principle components vs variance plot

In our observation, by considering the first 8 Principle components we got maximum proportion of variance. We train a model by considering the first 8 principle components in the projected space for 40,000 domain samples (10,000 random samples from each type i.e Matsnu, Gozi, Suppobox, Benign). According to our observations there is a 4% drop in the accuracy by considering top 8 principle components as shown in Table 4. In our observation, C5.0 is best among all other classifiers with low False Positive Rate (FPR) and False Negative Rate (FNR) for this PCA dataset. We found a large number of domains from Gozi, Matsnu and Suppobox families are misclassified as Benign i.e less significant principle components are impacting the decision stumps of ensemble models for a perfect classification. Number of decision stumps increases in this case as we run classification on projected data. So, models sizes are drastically increased (almost doubled) in this case when compared with standard 15 feature model.

Table 3. Results obtained after applying various classifier models on 8 feature dataset

Algorithm	Accuracy	Kappa	Sensitivity	Specificity
Naive Bayesian	0.8142	0.7522	0.8718(B), 0.8682(M), 0.9958(G), 0.5210(S)	0.9962(B), 0.9619(M), 0.8138(G), 0.9802(S)
KNN	0.9656	0.9541	0.8720(B), 0.9968(M), 0.9962(G), 0.9972(S)	0.9992(B), 0.9970(M), 0.9595(G), 0.9983(S)
CART	0.8230	0.7640	0.6790(B), 0.7588(M) 0.9200(G), 0.9343(S)	0.9997(B) 0.9414(M) 0.9137(G), 0.9092(S)
GBM	0.9788	0.9717	0.9187(B), 0.9965(M), 0.9998(G), 1.00(S)	1.00(B) 0.9932(M), 0.9786(G), 0.9999(S)
Random Forest	0.9796	0.9728	0.9255(B), 0.9948(M), 0.9982(G), 1.00(S)	0.9999(B), 0.9971(M), 0.9758(G), 1.00(S)
C 5.0	0.9777	0.9702	0.9117(B), 0.9990(M), 1.00(G), 1.00(S)	0.9999(B), 0.9892(M), 0.9812(G), 1.00(S)
J48	0.9758	0.9677	0.9110(B) 0.9930(M), 0.9990(G), 1.0000(S)	0.9998(B) 0.9920(M), 0.9761(G), 0.9998(S)

Table 4. Results obtained for various classifier models after PCA

Algorithm	Accuracy	Kappa	Sensitivity	Specificity
Naive Bayesian	0.8155	0.7540	0.8640(B), 0.7896(M), 0.8358(G), 0.7726(S)	0.9405(B), 0.9288(M), 0.9471(G), 0.9376(S)
KNN	0.7269	0.6359	0.6324(B), 0.6882(M), 0.8182(G), 0.7688(S)	0.9147(B), 0.8951(M), 0.9190(G), 0.9071(S)
CART	0.7231	0.6308	0.9402(B), 0.8612(M), 0.6774(G), 0.4136(S)	0.9053(B), 0.7927(M), 0.9666(G) 0.9661(S)
GBM	0.9156	0.8874	0.9408(B), 0.8784(M), 0.9686(G), 0.8744(S)	0.9880(B), 0.9569(M), 0.9863(G), 0.9561(S)
Random Forest	0.9365	0.9153	0.9550(B), 0.9152(M), 0.9816(G), 0.8942(S)	0.9922(B), 0.9630(M), 0.9893(G), 0.9709(S)
C 5.0	0.9376	0.9168	0.9594(B), 0.9316(M), 0.9784(G), 0.8810(S)	0.9903(B), 0.9594(M), 0.9907(G), 0.9764(S)
J48	0.9122	0.8829	0.9382(B), 0.8634(M), 0.9700(G), 0.8770(S)	0.9884(B), 0.9559(M), 0.9845(G), 0.9541(S)

4.4 Experiment-4

We apply one of the nonlinear dimensionality reduction technique called the Diffusion Map. This technique helps to understand the underlying geometric structure of high dimensional data as well as reduce dimensions by methodically capturing non-linear relations between the original dimensions [31]. We consider 4800 domain samples (random 1200 samples from each type i.e Matsnu, Gozi, Suppobox, Benign with 15 features). We apply Diffusion Map on these 4800 samples data set with different alpha values. In addition, we applied K-means (unsupervised clustering technique) at both normal space and diffusion space as shown in Table 5. We consider very low alpha to touch all the points. Based on these results, it is evident that there is no underlying structure for this dataset as illustrated in Figs. 8. and 9 respectively.

Cluster in 3D with alpha = 0.005

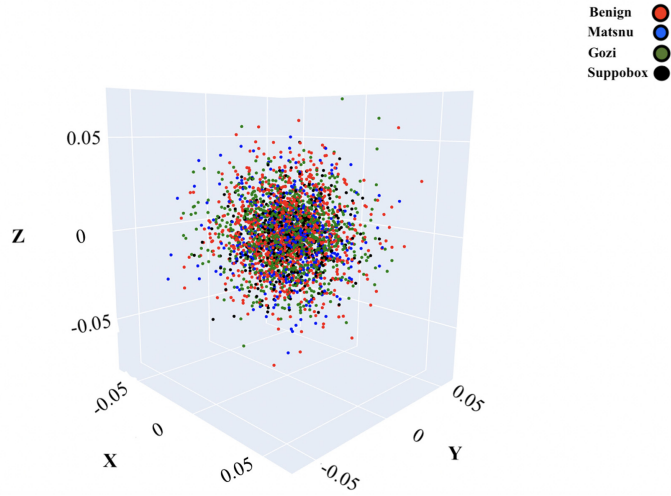


Fig. 8. Diffusion map plot with alpha = 0.005

K-Means Cluster in 3D

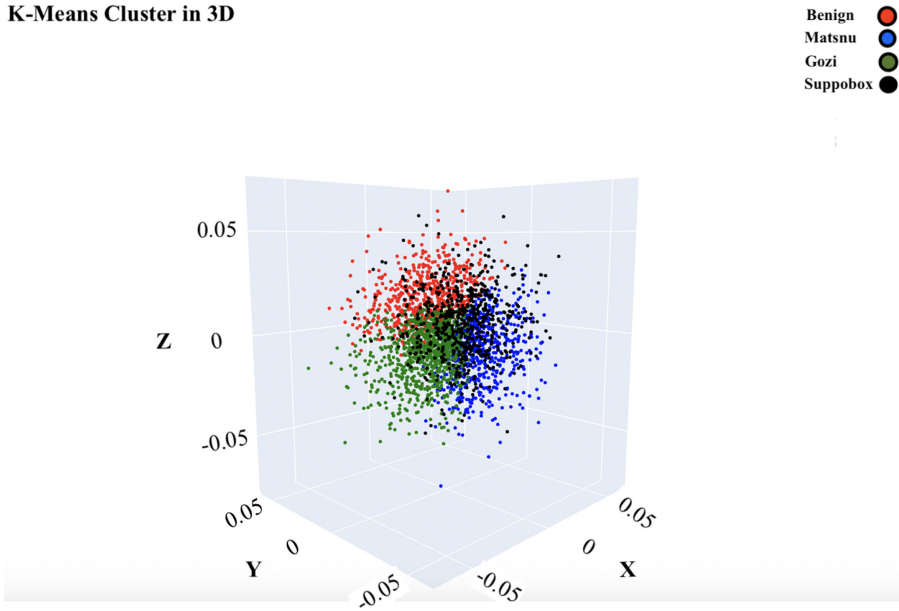


Fig. 9. K-means on diffusion map data (alpha = 0.005)

Table 5. Results obtained for various experiments with diffusion map

S. No.	Case	Dimension	Alpha	Benign	Matsnu	Gozi	Suppobox	Cluster-1	Cluster-2	Cluster-3	Cluster-4
1	K-Means in original Space	13	NA	1200	1200	1200	1200	956	337	1763	1744
2	K-Means in Diffusion Space	13	0.01	1200	1200	1200	1200	382	3782	371	265
3	K-Means in Diffusion Space	3	0.01	1200	1200	1200	1200	3765	352	340	343
4	K-Means in Diffusion Space	3	0.005	1200	1200	1200	1200	669	665	2811	655
5	K-Means in Diffusion Space	3	0.001	1200	1200	1200	1200	984	932	830	2054
6	K-Means in Diffusion Space	13	0.001	1200	1200	1200	1200	709	763	2572	756

4.5 Experiment-5

In this experiment, we test the robustness of our model. We used CTGAN, which is a GAN based synthesizer to generate data with high precision [32]. Basically, CTGAN is an evolved version of TGAN (open source project of Data to AI lab from MIT) and it can effectively work by generating synthetic data from numerical, categorical columns as well. We choose CTGAN over TGAN to generate synthetic data because of these reasons:

- CTGAN uses more advanced Variational Gaussian Mixture Model to detect continuous column modes.
- TGAN uses LSTM to generate synthetic column data. CTGAN uses more efficient, fully-connected networks.
- In CTGAN, we have a conditional generator to resample training data in order to prevent the model from collapsing into discreet columns.

We generate 30000 synthetic data samples (10000 samples from each DGA family) from the original 15 feature dataset. We combine these 30,000 synthetic domain name samples with 4000 legitimate domain names samples to build a new test dataset. We test the robustness of our previously built C5.0 model by passing this new test dataset as shown in Fig. 10. Our model did a decent work by classifying malware and benign domain samples with 0.9503 accuracy. In our experiment, out of 30000 synthetically generated malicious domain names, 1351 are classified as benign.

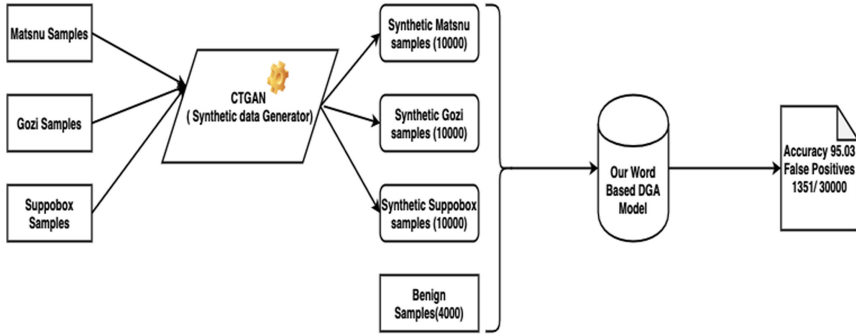


Fig. 10. Generating synthetic data for DGA families using CTGAN

5 Conclusion and Future Scope

In today’s modern cybersecurity era, constant change in the IP addresses and the continuous generation of thousands of DGA is a serious concern for many security researchers and malware investigators. Especially, these word-list based DGA are so agile and configurable in nature that has become a popular practice for many APT groups across the globe. In this paper, we addressed this problem of detecting word-list based DGA domain names (Matsnu, Gozi and Suppobox) using ensemble models with near real-time detection by considering both lexical and network level features. We also applied different linear and non-linear dimensionality reduction techniques to understand the underlying structure of our data. In addition, we used CTGAN to generate synthetic data (test data) to measure the robustness of our model. Although we considered significant families of word list based DGA, still we need to cover many emerging DGA families. So, possible future work is to extend this approach for next generation DGA families and building an AI component for word-based DGA detection that can be incorporated with traditional detection mechanisms. Similarly, we can also use GAN to generate synthetic data for future DGA families and that data can be used for training next generation robust malware/botnet detection models.

References

1. Chen, X., et al.: Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In: IEEE International Conference on Dependable Systems and Networks with FTCS and DCC (DSN), pp. 177–186. IEEE (2008)
2. Sai Charan, P.V., Gireesh Kumar, T., Mohan Anand, P.: Advance persistent threat detection using long short term memory (LSTM) neural networks. In: Somani, A.K., Ramakrishna, S., Chaudhary, A., Choudhary, C., Agarwal, B. (eds.) ICETCE 2019. CCIS, vol. 985, pp. 45–54. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-8300-7_5

3. Sood, A.K., Zeadally, S.: A taxonomy of domain-generation algorithms. *IEEE Secur. Privacy* **14**(4), 46–53 (2016)
4. Kumar, A., Gupta, M., Kumar, G., Handa, A., Kumar, N., Shukla, S.K.: A review: malware analysis work at IIT Kanpur. In: Shukla, S.K., Agrawal, M. (eds.) *Cyber Security in India*. ID, vol. 4, pp. 39–48. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-1675-7_5
5. Schiavoni, S., Maggi, F., Cavallaro, L., Zanero, S.: Phoenix: DGA-based botnet tracking and intelligence. In: Dietrich, S. (ed.) *DIMVA 2014*. LNCS, vol. 8550, pp. 192–211. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08509-8_11
6. Royal, P.: Analysis of the kraken botnet. Damballa, 9 April 2008
7. Shin, S., Gu, G.: Conficker and beyond: a large-scale empirical study. In: *Proceedings of the 26th Annual Computer Security Applications Conference*, pp. 151–160 (2010)
8. Mohaisen, A., Alrawi, O.: Unveiling zeus: automated classification of malware samples. In: *Proceedings of the 22nd International Conference on World Wide Web*, pp. 829–832 (2013)
9. Brahara, B., Syamsuar, D., Kunang, Y.N.: Analysis of malware DNS attack on the network using domain name system indicators. *J. Inf. Syst. Inform.* **2**(1), 131–153 (2020)
10. Anand, P.M., Kumar, T.G., Charan, P.S.: An Ensemble approach for algorithmically generated domain name detection using statistical and lexical analysis. *Procedia Comput. Sci.* **171**, 1129–1136 (2020)
11. Berman, D.S., et al.: DGA CapsNet: 1D application of capsule networks to DGA detection. *Information* **10**(5), 157 (2019)
12. Matrosov, A., Rodionov, E.: Defeating x64: modern trends of kernel-mode rootkits (2011). <https://www.eset.com/fileadmin/ezet/US/resources/docs/white-papers/white-papers-defeating-x-64-modern-trends-of-kernel-mode-rootkits.pdf>. Accessed 21 Oct 2011
13. Matsnu-DGA. <https://www.securityweek.com/new-variant-matsnu-trojan-uses-configurable-dg>. Accessed 15 June 2020
14. Fu, Y.: Using botnet technologies to counteract network traffic analysis (2017)
15. Yadav, S., Reddy, A.K.K., Reddy, A.N., Ranjan, S.: Detecting algorithmically generated malicious domain names. In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, pp. 48–61 (2010)
16. Da Luz, P.M.: Botnet detection using passive DNS. Radboud University, Nijmegen, The Netherlands (2014)
17. Selvi, J., Rodríguez, R.J., Soria-Olivas, E.: Detection of algorithmically generated malicious domain names using masked N-grams. *Expert Syst. Appl.* **124**, 156–163 (2019)
18. Plohmann, D., Yakdan, K., Klatt, M., Bader, J., Gerhards-Padilla, E.: A comprehensive measurement study of domain generating malware. In: *25th USENIX Security Symposium (USENIX Security 16)*, pp. 263–278 (2016)
19. Curtin, R.R., Gardner, A.B., Grzonkowski, S., Klymenov, A., Mosquera, A.: Detecting DGA domains with recurrent neural networks and side information. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pp. 1–10 (2019)
20. Yang, L., et al.: Detecting word-based algorithmically generated domains using semantic analysis. *Symmetry* **11**(2), 176 (2019)
21. Woodbridge, J., Anderson, H.S., Ahuja, A., Grant, D.: Predicting domain generation algorithms with long short-term memory networks. *arXiv preprint arXiv:1611.00791* (2016)

22. Choi, H., Lee, H., Kim, H.: BotGAD: detecting botnets by capturing group activities in network traffic. In: Proceedings of the Fourth International ICST Conference on COMMunication System softWARE and middlewaRE, pp. 1–8 (2009)
23. Abbink, J., Doerr, C.: Popularity-based detection of domain generation algorithms. In: Proceedings of the 12th International Conference on Availability, Reliability and Security, pp. 1–8 (2017)
24. Word Ninja. <https://github.com/keredson/wordninja>. Accessed 15 June 2020
25. whois 0.9.6. <https://pypi.org/project/whois>. Accessed 15 June 2020
26. Mixed Naive Bayes. <https://pypi.org/project/mixed-naive-bayes>. Accessed 15 June 2020
27. Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. *Chemometr. Intell. Lab. Syst.* **2**(1–3), 37–52 (1987)
28. De la Porte, J., Herbst, B.M., Hereman, W., Van Der Walt, S.J.: An introduction to diffusion maps. In: Proceedings of the 19th Symposium of the Pattern Recognition Association of South Africa (PRASA 2008), Cape Town, South Africa, pp. 15–25 (2008)
29. Zheng, T., Salganik, M.J., Gelman, A.: How many people do you know in prison? Using overdispersion in count data to estimate social structure in networks. *J. Am. Stat. Assoc.* **101**(474), 409–423 (2006)
30. Yan, K., Zhang, D.: Feature selection and analysis on correlated gas sensor data with recursive feature elimination. *Sens. Actuat. B: Chem.* **212**, 353–363 (2015)
31. Diffusion Map for Manifold Learning. <https://www.kdnuggets.com/2020/03/diffusion-map-manifold-learning-theory-implementation.html>. Accessed 15 June 2020
32. Xu, L., Skoularidou, M., Cuesta-Infante, A., Veeramachaneni, K.: Modeling tabular data using conditional gan. In: Advances in Neural Information Processing Systems, pp. 7335–7345 (2019)

Credentials



Distance-Bounding, Privacy-Preserving Attribute-Based Credentials

Daniel Bosk¹(✉), Simon Bouget², and Sonja Buchegger¹

¹ KTH Royal Institute of Technology, Stockholm, Sweden
daniel@bosk.se, buc@kth.se

² RISE Research Institutes of Sweden, Stockholm, Sweden
simon.bouget@ri.se

Abstract. Distance-bounding anonymous credentials could be used for any location proofs that do not need to identify the prover and thus could make even notoriously invasive mechanisms such as location-based services privacy-preserving. There is, however, no secure distance-bounding protocol for general *attribute-based* anonymous credentials. Brands and Chaum's (EUROCRYPT'93) protocol combining distance-bounding and Schnorr identification comes close, but does not fulfill the requirements of modern distance-bounding protocols. For that, we need a secure distance-bounding zero-knowledge proof-of-knowledge resisting mafia fraud, distance fraud, distance hijacking and terrorist fraud.

Our approach is another attempt toward combining distance bounding and Schnorr to construct a distance-bounding zero-knowledge proof-of-knowledge. We construct such a protocol and prove it secure in the (extended) DFKO model for distance bounding. We also performed a symbolic verification of security properties needed for resisting these attacks, implemented in Tamarin.

Encouraged by results from Singh et al. (NDSS'19), we take advantage of lessened constraints on how much can be sent in the fast phase of the distance-bounding protocol and achieve a more efficient protocol. We also provide a version that does not rely on being able to send more than one bit at a time which yields the same properties except for (full) terrorist fraud resistance.

D. Bosk—Thanks to Sébastien Gambs (UQAM), Cristina Onete (Univ. Limoges) and Douglas Wikström (KTH) for valuable discussions. Thanks to Mats Näslund (FRA, KTH) for reading the draft and pointing out several mistakes. Part of the work done while visiting the WIDE team in Inria/CNRS/IRISA/Univ. Rennes. Supported by the Swedish Foundation for Strategic Research grant SSF FFL09-0086.

S. Bouget—Supported by the funding for H2020 project CONCORDIA (Grant Agreement No. 830927). Part of the work done while at KTH, funded by the Swedish Foundation for Strategic Research, grant SSF FFL09-0086.

S. Buchegger—Supported by the Swedish Foundation for Strategic Research grant SSF FFL09-0086.

1 Introduction

Distance Bounding Protocols. Distance bounding (DB) [1] protocols were first suggested by [1] to prevent relay attacks in contactless communications in which the adversary forwards a communication between a prover and a possibly far-away verifier to authenticate. These attacks cannot be prevented by cryptographic means as they are independent of the semantics of the messages exchanged. DB protocols precisely enable the verifier to estimate an upper bound on his distance to the prover by measuring the time-of-flight of challenge-response messages (or rounds) exchanged during time-critical phases. Time critical phases are complemented by slow phases during which the time is not taken into account. At the end of a DB protocol, the verifier should be able to determine whether the prover is legitimate *and* in his vicinity. In this sense, DB protocols combine the classical properties of authentication protocols with the possibility of verifying the physical proximity.

There are four adversaries for DB protocols proposed in the literature, each of which tries to commit a type of fraud. These can be summarized as follows:

- Distance fraud (DF) [1]: a legitimate but malicious prover wants to fool the verifier on the distance between them.
- Mafia fraud (MF) [2]: the adversary illegitimately authenticates using a, possibly honest, prover who is far away from the verifier. (Also known as relaying attack or man-in-the-middle attack.)
- Terrorist fraud (TF) [3]: a legitimate, but malicious, prover helps an accomplice, who is close to the verifier, to authenticate. TF resistance is a very strong property; it implies that if the accomplice succeeds (with non-negligible probability) he will learn the prover’s secret key¹.
- Distance hijacking (DH) [4]: similar to DF, the malicious prover is far away but uses an unsuspecting honest prover close to the verifier to pass as being close. (This is different from MF in that the honest prover actually tries to authenticate to the verifier, but the malicious prover hijacks the channel at some point(s) during the protocol.)

The majority of existing DB protocols are symmetric and thus require an honest verifier. In this context, it does not make sense to protect against the verifier as he can easily impersonate the prover since he has knowledge of their shared secret key. This works for scenarios like key-less entry protocols for cars, where the car and keyfob can share a key. It does, however, not work for ubiquitous DB authentication where the prover and verifier may not have met before. We are interested in that latter scenario and will work in the public-key setting; with a *malicious verifier* who will potentially try to *impersonate the prover* after successful verification. Unfortunately, there has been less work done in the domain of asymmetric (or public-key) DB protocols.

¹ This means that even things like functional encryption will not help.

This Paper. Our work is in the area of DB public-key protocols and presents the first DB zero-knowledge proofs of knowledge (ZKPKs) for discrete logarithms. The public-key setting allows anyone to act as a verifier, unlike in the shared-key scenario where the verifier must be trusted. The zero-knowledge (ZK) property also enables a variety of privacy properties, such as anonymity. With our contribution, Alice can now prove more complex statements to Bob, e.g.: that she *simultaneously* is close to a specific car, has a valid driver’s license, is older than 18 (without revealing her actual age), has the relevant car-sharing subscription which is not “double spent” by someone else at the same time [e.g., 5], and Bob can be sure that Alice is not just relaying messages from her older sister Carol or MF relaying the subscription from some unsuspecting stranger also waiting in the parking lot.

Related Work. Brands and Chaum [1] were the first to adapt the Schnorr protocol to achieve distance-bounding. Unfortunately, their approach was shown to be vulnerable [4, 6] to DH. Furthermore, it is also vulnerable to TF. (However, these frauds were defined many years after their paper.)

The Bussard-Bagga [7] protocol aims to overcome these vulnerabilities and also uses Schnorr, but slightly differently. They generate a session key and encrypt the private key with this session key. Then the prover commits to each bit of the session key and each bit of the ciphertext. During the fast phase, the verifier requests some bits from the session key and some from the ciphertext. Terrorist fraud is based on that knowing both session key and ciphertext one can compute the private key. The prover then opens the commits, the verifier verifies them. The prover and verifier run a proof of knowledge (PK) that ties the commits to the private key. However, a malicious prover can reduce all the N challenges in the fast phase to 1. This allows both distance and terrorist fraud [8].

There exists one protocol in the literature that provides proofs of proximity of knowledge with most of the desired properties described previously, ProProx [9]². ProProx is secure against a malicious verifier, but it provides a PPK protocol for quadratic residues (i.e., protocols of the form $\text{PPK}\{(\alpha) : a = \alpha^2\}$), while in our context, we need a PPK protocol for discrete logarithms (i.e., $\text{PPK}\{(\alpha) : a = g^\alpha\}$)—as many schemes for attribute-based credentials relies on discrete logarithms.

Contributions. We make three contributions. First, we provide a version of the Schnorr protocol [10] that is MF, TF, DH and DF resistant in the Dürholz-Fischlin-Kasper-Onete model (DFKO model) [11–13]³. This version is based on the assumption that we can transmit all bits at once, not bit-by-bit as normally done for DB. This assumption is motivated by the results of Singh, Leu, and Capkun [14], who show that it is possible to do secure DB while sending more than one bit at once (with some restrictions).

For the assumption about sending all bits at once to hold, Singh, Leu, and Capkun [14] requires that two devices share a symmetric key k . Our second

² Note that [9] uses the abbreviation PoPoK, we prefer PPK for shorter notation.

³ One of the two competing formal models for DB protocols.

contribution removes that requirement via a public-key infrastructure (PKI) and authenticated key-exchange (AKE) [15] scheme. The PKI and AKE allows any device in the PKI to convince a verifier that the prover is indeed a device that is part of the PKI. This replaces a static key k shared with everyone by a unique key for each signed by a certificate authority (CA) of the PKI. (This also allows for revocation.) This solution relies on our first contribution.

Our last contribution is a classic version of the protocol, one which uses bit-by-bit transmissions as traditionally done with DB protocols. We achieve all properties except for (full) TF resistance. Though our earlier contributions are more efficient, we provide this solution in case the solution of Singh, Leu, and Capkun [14] is not available.

We also formally verified both the all-bits-at-once and the bit-by-bit protocols. The detailed discussion and proofs can be found in the full version of the paper.

The first (and second) contribution enables us to do distance-bounding privacy-preserving attribute-based credentials based on discrete logarithms, e.g., [16] and [17]. We can do this by simply replacing their use of the Schnorr protocol for ZKPKs with our protocol in this paper. To our knowledge, no other distance-bounding protocol achieves this. The third contribution (bit-by-bit version) also enables us to do the same, just without (full) TF resistance.

Outline. The paper is organized as follows. Section 2 introduces the preliminaries of ZKPK and DB. Section 3 introduces the first version, where we assume that we can send all bits at the same time. Section 4 introduces the PKI and AKE to use the first version of the protocol in practice. Section 5 describes the bit-by-bit version of the protocol. In Sect. 6, we discuss the performance implications of making the ZKPK of the Schnorr identification scheme [10] distance bounding. Section 7 concludes the paper.

2 Preliminaries

In this section we introduce some notation and the formal models that we will use.

2.1 Zero-Knowledge Proofs of Knowledge

We will use the notation introduced by Camenisch and Stadler [18]:

$$\text{PK}\{(\alpha, \beta, \gamma) : y = g^\alpha h^\beta \wedge y' = \hat{g}^\gamma\}, \quad (1)$$

is a PK protocol where the prover proves knowledge of the discrete logarithms α, β, γ ensuring that y, y' are of the form $y = g^\alpha h^\beta$ and $y' = \hat{g}^\gamma$, respectively. Greek letters are secret, known only to the prover, and all other letters are public.

Instantiations of the ZKPK Protocols. For the convenience of the reader, we summarize (from [16]) how to instantiate general ZKPKs for discrete logarithms based on the Schnorr identification scheme [10], which, in its original form, can be written as $\text{PK}\{(\alpha) : A = g^\alpha\}$. The generalized form would be

$$\text{PK}\left\{(\alpha_1, \dots, \alpha_n) : A = \prod_{i=1}^n g_i^{\alpha_i}\right\}.$$

This could equivalently be written as

$$\langle \text{PK.Prove}(\{g_i\}_i, q, A, \{\alpha_i\}_i); \text{PK.Verify}(\{g_i\}_i, q, A) \rangle,$$

where PK.Prove is run by the prover and PK.Verify is run by the verifier. PK.Verify outputs accept (\top) or reject (\perp). We give an instance of PK.Prove and PK.Verify in Fig. 1. We also note that

$$\text{PK}\{(\alpha, \beta) : A = g^\alpha \wedge B = g^\beta\} = \text{PK}\{(\alpha) : A = g^\alpha\} \wedge \text{PK}\{(\beta) : B = g^\beta\},$$

where the challenge c must be the same in both sub-protocols.

$\text{PK.Prove}(\{g_i\}_{i=1}^n, q, \{\alpha_i\}_{i=1}^n):$	$\text{PK.Verify}(\{g_i\}_{i=1}^n, q, A):$
$\forall i, 1 \leq i \leq n: \rho_i \xleftarrow{c} \mathbb{Z}_q$	
$R \leftarrow \prod_{i=1}^n g_i^{\rho_i}$	\xrightarrow{R}
	$\xleftarrow{c} c \xleftarrow{c} \{0, 1\}^k$
$\forall i: s_i \leftarrow \rho_i - c\alpha_i \pmod q$	$\xrightarrow{\{s_i\}_{i=1}^n}$
	$R \stackrel{?}{=} A^c \prod_{i=1}^n g^{s_i}$

Fig. 1. $\text{PK}\{(\alpha_1, \dots, \alpha_n) : A = \prod_{i=1}^n g_i^{\alpha_i}\}$ using the Schnorr identification scheme [16].

The Schnorr protocol in itself only provides honest-verifier zero-knowledge. To achieve malicious-verifier zero-knowledge, we have to choose k (size of the challenge, see Fig. 1) logarithmic in the security parameter λ and repeat the protocol sufficiently many times (also logarithmic in the security parameter) [16].

2.2 Distance Bounding Security Definitions

There are two lines of attempts at formalizing the above properties: one by Boureau, Mitrokotsa, and Vaudenay [19] and another by Dürholz, Fischlin, Kasper, and Onete [11]. We will use the latter one (the DFKO model) with extensions by Fischlin and Onete [12] and Avoine, Bultel, Gambs, *et al.* [13] (definitions in Sect. 2.2).

As summarized in by Avoine, Bultel, Gambs, *et al.* [13], there are three types of sessions for an adversary:

Prover–Verifier. The adversary observes an honest execution.

Prover–Adversary. The adversary runs the protocol as a verifier with an honest prover.

Adversary–Verifier. The adversary runs the protocol as a, potentially malicious, prover with an honest verifier.

Each session is associated with a unique identifier sid . Each adversary is defined in terms of computational resources, time t ; number of prover–verifier sessions observed, q_{obs} ; number of prover–adversary sessions initiated, q_P , and number of adversary–verifier sessions initiated, q_V .

The DFKO model uses an abstract clock **marker** which is strictly increasing and orders everyone’s actions. We let $\Pi_{sid}[i, \dots, j]$ denote the sequence of messages (m_i, \dots, m_j) exchanged during the session sid . We let $\text{marker}(sid, i)$ return the “time” for when the i th message was sent in the session sid .

A session consists of three phases: a setup phase, a time-critical phase and a verification phase. Both setup and verification can be run slowly, but the time-critical phase must be run as fast as possible.

Mafia Fraud. We will now define the conditions for MF and DBMF resistance. A *tainted session* is a session in which the adversary lost because the verifier detected the attempt at cheating, e.g., the timing is too long because the adversary is out of range. The following definition captures a pure relay and a verifier is assumed to detect such a relay.

Definition 1 (Tainted session, MF [11]). *A time-critical round*

$$\Pi_{sid}[k, k+1] = (m_k, m_{k+1})$$

of an adversary–verifier session sid , where m_k is sent by the verifier, is tainted by the round

$$\Pi_{sid'}[k', k'+1] = (m_{k'}, m_{k'+1})$$

of a prover–adversary session sid' if

$$\begin{aligned} (m_k, m_{k+1}) &= (m_{k'}, m_{k'+1}) \\ \text{marker}(sid, k) &< \text{marker}(sid', k') \\ \text{marker}(sid, k+1) &> \text{marker}(sid', k'+1). \end{aligned}$$

We will now define what it means for a protocol to be MF resistant.

Definition 2 (MF resistance [13]). *For a DB authentication scheme DB, a $(t, q_{\text{obs}}, q_P, q_V)$ -MF adversary \mathcal{A} wins against DB if the verifier accepts \mathcal{A} in one of the q_V adversary–verifier sessions sid , which does not have any critical phase tainted by a prover–adversary session sid' . The protocol DB is MF resistant if the probability $\text{Adv}_{\text{DB}}^{\text{MF}}(\mathcal{A})$ that \mathcal{A} wins is negligible in the security parameter.*

Terrorist Fraud. Terrorist fraud is similar to MF, the difference is that during TF the prover cooperates with the adversary to make the verifier accept. This means that we must disallow relay *scheduling*: with Definition 1, the prover and adversary could relay a function of every answer, then the adversary could just apply the inverse function before sending the answer to the verifier.

Definition 3 (Tainted session, TF (strSimTF) [12]). *A time-critical round*

$$\Pi_{sid} [k, \dots, k + 2l - 1] = (m_k, \dots, m_{k+2l-1}),$$

for $l \geq 1$, of an adversary–verifier session sid , where m_k is sent by the verifier, is tainted by the round

$$\Pi_{sid'} [k, \dots, k + 2l - 1] = (m'_k, \dots, m'_{k+2l-1})$$

of a prover–adversary session sid' if

$$\begin{aligned} \text{marker}(sid, k + 2i) &< \text{marker}(sid', k + 2i) \\ \text{marker}(sid, k + 2i + 1) &> \text{marker}(sid', k + 2i + 1). \end{aligned}$$

Then we have the following definition of security.

Definition 4 (SimTF security [11]). *Let DB be an authentication scheme for parameters $(t_{\max}, T_{\max}, E_{\max}, N_c)$. Let \mathcal{A} be a $(t, q_V, q_{\mathcal{F}})$ -SimTF adversary, \mathcal{S} an algorithm with runtime $t_{\mathcal{S}}$ and \mathcal{F} an algorithm with runtime $t_{\mathcal{F}}$. Let*

$$\text{Adv}_{\text{DB}}^{\text{TF}}(\mathcal{A}, \mathcal{S}, \mathcal{F}) = p_{\mathcal{A}} - p_{\mathcal{S}},$$

where $p_{\mathcal{A}}$ is the probability that the verifier V accepts in one of the q_V adversary–verifier sessions sid such that at T_{\max} time-critical phases of sid are tainted and $p_{\mathcal{S}}$ is the probability that, given the state of \mathcal{A} , \mathcal{S} authenticates to the verifier V in one of q_V subsequent executions.

Distance Hijacking and Distance Fraud. The final properties are DH and DF. The following definition of DH also captures DF attacks [13]. We first define what the adversary can and cannot do by defining a tainted session for DH.

The intuition is that an adversary (malicious, far-away prover) must commit to an action: either respond with an own message, which is determined before seeing the challenge, or to prompt a nearby prover to respond (with a message chosen by the adversary). Technically, this adversary can do more than a DH adversary normally can. Avoine, Bultel, Gams, *et al.* [13] captured this by two dummy sessions sid_{Commit} and sid_{Prompt} . The adversary commits to his action by “sending” $(sid, k + 1, m'_{k+1})$ in dummy session sid_{Commit} before receiving the challenge in message k in session sid (making message $k + 1$ in sid his response to the challenge). If $m'_{k+1} = \text{Prompt}$, then he must also “send” a message in dummy session sid_{Prompt} , which is the message the nearby prover will send.

Definition 5 (Tainted session, DH [13]). *A time-critical round $\Pi_{sid}[k,]k + 1 = (m_k, m_{k+1})$, for some $k \geq 1$ and m_k send by the verifier, of an adversary-verifier session sid is tainted if one of the following conditions holds.*

- *The maximal j with $\Pi_{sid_{\text{Commit}}}[j] = (sid, k + 1, m'_{k+1})$ for $m'_{k+1} \neq \text{Prompt}$ and $\text{marker}(sid, k) > \text{marker}(sid_{\text{Commit}}, j)$ satisfies $m'_{k+1} \neq m_{k+1}$ (or no such j exists).*
- *The maximal j with $\Pi_{sid_{\text{Commit}}}[j] = (sid, k + 1, m'_{k+1})$ for $m'_{k+1} = \text{Prompt}$ satisfies $m_{k+1} \neq m_{k+1}^{\text{Prompt}}$, in which m_{k+1}^{Prompt} denotes the message m_{k+1} in sid_{Prompt} .*

To win, the adversary must convince a verifier that he is close when he actually is not and without tainting any of the sessions.

Definition 6 (DH resistance [13]). *For a DB authentication scheme DB with DB threshold t_{\max} , a $(t, q_P, q_V, q_{\text{obs}})$ -DH adversary \mathcal{A} wins against DB if the verifier accepts \mathcal{A} in one of the q_V adversary-verifier sessions without any critical round being tainted. Thus, DH resistance is defined as the probability $\text{Adv}_{\text{DB}}^{\text{DH}}(\mathcal{A})$ that \mathcal{A} wins the game.*

3 Distance-Bounding ZKPK for Discrete Logarithms

We will now introduce a DB protocol which is a ZKPK for discrete logarithms. Our protocol is an adaptation of the Schnorr identification scheme [10], albeit different from that of Brands and Chaum [1] in the original DB paper and that of Bussard and Bagga [7]. We propose another way to turn the Schnorr protocol into a public-key DB protocol which is a ZKPK that is secure against MF, DF, DH and TF. This yields strong privacy properties and protection against a malicious, impersonating verifier.

We will provide three versions of the protocol. In the first one (Sect. 3.1), we will assume that the prover and verifier can send as many bits in parallel as they like. This is a reasonable assumption thanks to the results of Singh, Leu, and Capkun [14]. Let us briefly explain why. An MF adversary can listen in on the communications. Each symbol of the communication is encoded using signal pulses. However, such an encoded symbol can usually be inferred *before* all pulses are received⁴. Prior to the contribution of Singh, Leu, and Capkun, the only countermeasure to this attack was to make the symbols short, i.e., to send one bit at a time. Singh, Leu, and Capkun [14] use a symmetric key, shared between prover and verifier, to scramble the pulses and hence the symbols. This way, the MF adversary cannot infer an incoming symbol in advance, and thus cannot start relaying the symbol in advance either. The prover and verifier, on the other hand, can do this since they have the symmetric key. Consequently, they must have mutual trust; so, by definition, DF cannot happen.

Next (Sect. 4), we will provide a PKI construction to make the shared-key requirement of Singh, Leu, and Capkun [14] more practical. This would allow

⁴ This is due to redundancy for the purpose of error correction.

any two devices to run our protocol as long as they are both part of the PKI. We use the first version of our protocol to ensure correctness of the PKI. This version also deals with DH and DF.

Finally (Sect. 5), we provide a classical bit-by-bit version of our protocol.

3.1 Reattempting a Distance-Bounding Schnorr Protocol

In this version, we assume that we can simply send as many bits in parallel as we like. This allows us to keep the Schnorr protocol almost as is.

We present the protocol in Fig. 2. The (cyclic) group with generator g and order q are system parameters. The private key α with public key $A = g^\alpha$ are generated once by the prover in the setup phase.

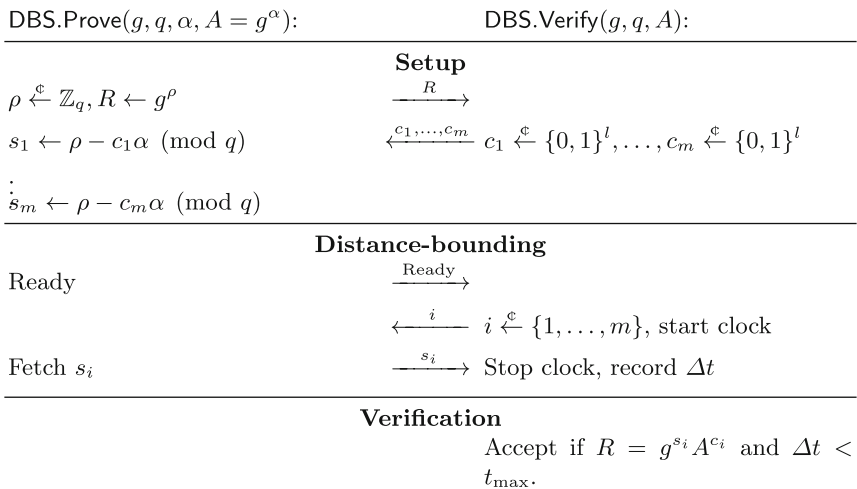


Fig. 2. One-round protocol instance of the DBS.Prove \leftrightarrow DBS.Verify protocol instantiating PK $\{(\alpha) : A = g^\alpha\}$. The protocol should be repeated n times to achieve the desired soundness and distance-bounding errors.

During one round, in the setup phase, the prover commits to a random nonce: more precisely he chooses $\rho \xleftarrow{c} \mathbb{Z}_q$ uniformly at random, computes $R \leftarrow g^\rho$ and sends R to the verifier. The verifier generates m challenges $c_1 \xleftarrow{c} \{0, 1\}^l, \dots, c_m \xleftarrow{c} \{0, 1\}^l$ (bit strings of length l) and sends them to the prover. The prover computes one response per challenge, $s_1 \leftarrow \rho - c_1 \alpha, \dots, s_m \leftarrow \rho - c_m \alpha$. This step is the main difference to the original Schnorr protocol: the verifier selects several challenges and the prover computes several responses—but still only use *one nonce*, ρ . This is needed for TF resistance. This is also different from the original Brands-Chaum protocol, in which the prover and verifier jointly construct *one* challenge with *one* response.

In the DB phase, the prover notifies the verifier that he has computed s_1, \dots, s_m . The verifier chooses one of the challenges, i , uniformly at random and sends it as a challenge to the prover and starts measuring the time of flight. The prover replies with s_i . The verifier stops the measurement and verifies that $R = g^{s_i} A^{c_i}$ and that the time of flight was within t_{\max} (determined from the allowed distance).

This protocol must be repeated n times.

3.2 Zero-Knowledge and Proof of Knowledge

The main difference between this protocol and the Schnorr protocol is that we have the prover compute responses for m different challenges, c_1, \dots, c_m . However, in the authentication step, the verifier chooses only one of those challenges. From the simulator's (and extractor's) perspective, there is no difference whether the verifier first chooses m and then chooses one of those m challenges, or if the verifier chooses the one challenge directly; the distribution is the same, $\frac{m}{n} \times \frac{1}{m} = \frac{1}{n}$. Therefore the standard proof [e.g., 20] for Schnorr as a malicious-verifier ZKPK protocol with soundness error 2^{-ln} still holds. (We note that l , which is also the length of the challenge bit string, must be logarithmic in the security parameter, λ , for a malicious verifier.)

3.3 MF, TF and DH Resistance

The intuition behind the protocol security is as follows. The prover must know the responses for all challenges to be sure to pass the DB phase. The reason for having several challenges but only one random nonce is that knowing at least two responses means learning the secret α . This gives us the incentives that prevent TF. (We also need it for MF and DF.) Bundling the authentication into the distance-bounding phase (difference from Brands-Chaum) prevents DH.

The proofs of Theorems 1 to 3 can be found in the full version of the paper.

Mafia Fraud. By Definition 1, any relaying will be detected by the verifier. This means that the adversary must use a different strategy. But we show that there is no such better strategy.

Theorem 1 (MF resistance). *Let \mathcal{A} be a $(t, q_{\text{obs}}, q_P, q_V)$ -MF adversary, then $\text{Adv}_{\text{DBS}}^{\text{MF}}(\mathcal{A}) = \frac{1}{m^n}$, where m is as in Sect. 3.1.*

Terrorist Fraud. The protocol is TF resistant. Indeed, if the malicious prover gives more than one response to the accomplice, the accomplice can compute the secret key and, thus, impersonate forever. And we show that any useful help will also allow the accomplice to impersonate forever.

Theorem 2 (TF resistance). *Let \mathcal{A} be a (t, q_V) -strSimTF adversary (aided by the prover), \mathcal{S} an algorithm with runtime $t_{\mathcal{S}}$. If the verifier's challenges are uniformly drawn, then $\text{Adv}_{\text{DB}}^{\text{TF}} = p_{\mathcal{A}} - p_{\mathcal{S}} < \left(\frac{2}{m}\right)^n$.*

Distance Hijacking. The protocol is also secure against distance hijacking due to the fact that the authenticating bit string is used during the DB phase. (Brands-Chaum used the challenge bit string, something that made their protocol vulnerable.) DH requires that the adversary finds a collision between his response and that of the honest prover. Thus the probability of success is equivalent to a collision for the responses for the chosen challenge—in each round.

Theorem 3 (DH/DF resistance). *Let \mathcal{A} be a (t, q_{obs}, q_P, q_V) -DH adversary, then $\text{Adv}_{DBS}^{DH}(\mathcal{A}) = \frac{1}{m^n}$.*

4 From Mutual Trust to PKI

All distance bounding protocols require hardware implementations [21]. We will use a *trusted* hardware implementation to overcome the limitation of Singh, Leu, and Capkun [14], i.e., that the prover and verifier must share a key. We will provide a PKI for the verifier to verify the trusted hardware. We will leverage this PKI to overcome the shared-key requirement to allow any two devices in this PKI to perform distance bounding. With a PKI based on discrete logarithms we will be able to perform a DB ZKPK which *simultaneously* proves (1) that the protocol is run by trusted hardware, (2) that trusted hardware is within proximity, (3) that some ZKPK statement about some discrete logarithms holds, and (4) that knowledge is within proximity.

We will now introduce an extended protocol, DBSHW, which represents the secure hardware implementation. Figure 3 presents an overview of our protocol. For simplicity of exposition, we present a PKI based on plain CL04 [17] signatures. However, this also works for CL02 [22], which is the base for the Direct Anonymous Attestation (DAA) [23] protocol used for Trusted Platform Module (TPM), so our protocol can be modified into a distance bounding DAA protocol.

In summary, the difference is as follows. DBS performs $\text{PK}\{(\alpha) : A = g^\alpha\}$ with a distance bound on the user’s behalf. DBSHW first runs a Diffie-Hellman key-exchange (DHKE) [24] to establish a shared key $k = \bar{g}^{\bar{x}\bar{y}}$ from $\bar{X} = \bar{g}^{\bar{x}}, \bar{Y} = \bar{g}^{\bar{y}}$, then performs

$\text{PK}\{(\alpha, \bar{x}, sk_P, \sigma) :$

$$A = g^\alpha \wedge \bar{X} = \bar{g}^{\bar{x}} \wedge \hat{X} = \text{VRF}_{sk_P}(\bar{X}) \wedge \text{Blind}(\sigma) = \text{Sign}_{sk}(\sigma)\},$$

also with a distance bound and $A = g^\alpha$ on behalf of the user; where sk_P is the prover’s private key, VRF is the verifiable random function (VRF) of Dodis and Yampolskiy [25], σ is a signature issued with signing key sk and Blind reblinds a signature. Proving knowledge of $\bar{X} = \bar{g}^{\bar{x}}$ and $\hat{X} = \bar{X}^{sk}$ prevents any man-in-the-middle (MITM) attack against the DHKE. Proving knowledge of a signature σ by a CA on the private key sk_P ensures that the prover is a device in the PKI of the CA.

Figure 3 provides an overview. For simplicity of the exposition, we present a version of DBSHW that allows the user to run $\text{PK}\{(\alpha) : A = g^\alpha\}$ (the Schnorr identification scheme) with a distance bound. This can, of course, be generalized to any ZKPK for discrete logarithms, for instance, CL04 [17].

DBSHW.Prove(g, q, α):	DBSHW.Verify(g, q, A):
Static $sk_P \in \mathbb{Z}_q, \sigma = (a, A, b, B, c)$	Static $pk = (\hat{q}, \hat{g}, \hat{g}, e, \hat{X}, \hat{Y}, \hat{Z})$
Pre-setup key-agreement	
$\bar{x} \stackrel{c}{\leftarrow} \mathbb{Z}_{\bar{q}}, \bar{X} \leftarrow \bar{g}^{\bar{x}},$	$\bar{y} \stackrel{c}{\leftarrow} \mathbb{Z}_{\bar{q}}$
$\hat{X} \leftarrow \text{VRF}_{sk}(\bar{X}) = \hat{g}^{1/(sk+\bar{X})}$	$\bar{y} \xrightarrow{\bar{X}, \hat{X}}$
$k \leftarrow \bar{Y}^{\bar{x}}$	$k \leftarrow \bar{X}^{\bar{y}}$
Load k into UWBPR	Load k into UWBPR
Signature rebinding	
$r_1, r_2 \stackrel{c}{\leftarrow} \mathbb{Z}_{\hat{q}}$	$\bar{\sigma} = (\bar{a}, \bar{A}, \bar{b}, \bar{B}, \bar{c})$
$\bar{a} = a^{r_1}, \bar{A} = A^{r_1}, \bar{b} = b^{r_1},$	$\bar{\sigma} \xrightarrow{(\bar{a}, \bar{A}, \bar{b}, \bar{B}, \bar{c})}$
$\bar{B} = B^{r_1}, \bar{c} = (c^{r_1})^{r_2}$	$v_x = e(\hat{X}, \bar{a}), v_{xy} = e(\hat{X}, \bar{b}),$
$v_x = e(\hat{X}, \bar{a}), v_{xy} = e(\hat{X}, \bar{b}),$	$v_x = e(\hat{X}, \bar{a}), v_{xy} = e(\hat{X}, \bar{b}),$
$V_{xy} = e(\hat{X}, \bar{B}), v_s = e(\hat{g}, \bar{c})$	$V_{xy} = e(\hat{X}, \bar{B}), v_s = e(\hat{g}, \bar{c})$
Setup	
$\rho \stackrel{c}{\leftarrow} \mathbb{Z}_q, \bar{\rho} \stackrel{c}{\leftarrow} \mathbb{Z}_{\bar{q}}, \hat{\rho}_{sk} \stackrel{c}{\leftarrow} \mathbb{Z}_{\hat{q}},$	$R, \bar{R}, \hat{R}, \hat{R}$
$\hat{\rho}_r \stackrel{c}{\leftarrow} \mathbb{Z}_{\hat{q}}, \hat{\rho}_{r'} \stackrel{c}{\leftarrow} \mathbb{Z}_{\hat{q}}$	$\xrightarrow{R, \bar{R}, \hat{R}, \hat{R}}$
$R \leftarrow g^\rho, \bar{R} \leftarrow \bar{g}^{\bar{\rho}},$	$\xrightarrow{c_1, \dots, c_k}$
$\hat{R} \leftarrow \hat{X}^{\hat{\rho}_{sk}}, \hat{R} \leftarrow v_s^{\hat{\rho}_{r'}} v_{xy}^{\hat{\rho}_{sk}} V_{xy}^{\hat{\rho}_r}$	$c_1 \stackrel{c}{\leftarrow} \{0, 1\}^l, \dots, c_m \stackrel{c}{\leftarrow} \{0, 1\}^l$
$\forall i \in \{1, \dots, m\}: $	$c_1 \stackrel{c}{\leftarrow} \{0, 1\}^l, \dots, c_m \stackrel{c}{\leftarrow} \{0, 1\}^l$
$s_i \leftarrow \rho - c_i \alpha \pmod{q}$	$s_i \leftarrow \rho - c_i \alpha \pmod{q}$
$\bar{s}_i \leftarrow \bar{\rho} - c_i \bar{x} \pmod{\bar{q}}$	$\bar{s}_i \leftarrow \bar{\rho} - c_i \bar{x} \pmod{\bar{q}}$
$\hat{s}_i^{(r')} \leftarrow \hat{\rho}_{r'} + c_i r' \pmod{\hat{q}}$	$\hat{s}_i^{(r')} \leftarrow \hat{\rho}_{r'} + c_i r' \pmod{\hat{q}}$
$\hat{s}_i^{(sk)} \leftarrow \hat{\rho}_{sk} - c_i sk \pmod{\hat{q}}$	$\hat{s}_i^{(sk)} \leftarrow \hat{\rho}_{sk} - c_i sk \pmod{\hat{q}}$
$\hat{s}_i^{(r)} \leftarrow \hat{\rho}_r - c_i r \pmod{\hat{q}}$	$\hat{s}_i^{(r)} \leftarrow \hat{\rho}_r - c_i r \pmod{\hat{q}}$
Distance-bounding	
Ready	$\xrightarrow{\text{Ready}}$
\xrightarrow{i}	$i \stackrel{c}{\leftarrow} \{1, \dots, m\}, \text{ start clock}$
Fetch $s_i, \bar{s}_i, \hat{s}_i^{(r')}, \hat{s}_i^{(sk)}, \hat{s}_i^{(r)}$	$\xrightarrow{s_i, \bar{s}_i, \hat{s}_i^{(r')}, \hat{s}_i^{(sk)}, \hat{s}_i^{(r)}} \text{ Stop clock, record } \Delta t$
Verification	
Accept if $\Delta t < t_{\max}$ and	
$R = A^{c_i} g^{s_i},$	
$\bar{R} = \bar{X}^{c_i} \bar{g}^{\bar{s}_i},$	
$\hat{R} = \left(\hat{g}^{\hat{X}} \right)^c \hat{X}^{\hat{s}_i^{(sk)}},$	
$v_x^{c_i} \hat{R} = v_s^{\hat{s}_i^{(r')}} v_{xy}^{\hat{s}_i^{(sk)}} V_{xy}^{\hat{s}_i^{(r)}},$	
$e(\bar{a}, \hat{Z}) = e(\hat{g}, \bar{A}),$	
$e(\bar{a}, \hat{Y}) = e(\hat{g}, \bar{b}),$	
$e(\bar{A}, \hat{Y}) = e(\hat{g}, \bar{B}).$	

Fig. 3. One round of the DBSHW protocol instantiating $\text{PK}\{\alpha\} : A = g^\alpha$. Every transmission uses UWBPR (except in the pre-setup phase). The protocol (setup, distance-bounding and verification phases) should be repeated n times to achieve the desired soundness and distance-bounding errors.

One-Time Initialization The prover's DBSHW device must be initialized with a (static) private key sk_P and a signature σ on that key by some CA with public key pk and signing key sk . These are computed as follows [cf. [17], Sect. 4.2].

We have public parameters $\hat{q}, \hat{g} \in \hat{\mathbb{G}}, \hat{\mathbf{g}} \in \hat{\mathbb{G}}_T, e$; where $e: \hat{\mathbb{G}} \times \hat{\mathbb{G}} \rightarrow \hat{\mathbb{G}}_T$ is a bilinear map, $\hat{g} \in \hat{\mathbb{G}}$ and $\hat{\mathbf{g}} \in \hat{\mathbb{G}}_T$ are generators of prime order \hat{q} (hence $e(\hat{g}, \hat{g}) = \hat{\mathbf{g}}$).

We let the CA's signing key be

$$sk = (\hat{x}, \hat{y}, \hat{z}) \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\hat{q}}^3 \quad \text{and} \quad pk = (\hat{q}, \hat{g}, \hat{\mathbf{g}}, e, \hat{X}, \hat{Y}, \hat{Z}),$$

be the corresponding public key, where

$$\hat{X} = \hat{g}^{\hat{x}}, \quad \hat{Y} = \hat{g}^{\hat{y}}, \quad \hat{Z} = \hat{g}^{\hat{z}}.$$

The prover chooses $sk_P \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\hat{q}}$ and computes a commitment $M = \hat{g}^{sk_P} Z^r$, where $r \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\hat{q}}$. Now the prover convinces the CA that he knows the key by running the following ZKPK protocol: $\text{PK}\{(sk_P, r) : M = \hat{g}^{sk_P} Z^r\}$.

Now the CA computes the signature $\sigma = (a, A, b, B, c)$, where

$$\alpha \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\hat{q}}, \quad a \leftarrow \hat{g}^\alpha, \quad A \leftarrow \hat{g}^{\hat{z}}, \quad b \leftarrow a^{\hat{y}}, \quad B \leftarrow A^{\hat{y}}, \quad c \leftarrow a^{\hat{x}} M^{\alpha \hat{x} \hat{y}}$$

and gives to the prover.

Pre-Setup Key-Agreement. The pre-setup phase consists of a key agreement, the prover and verifier must agree on a shared key k to use for the UWBPR protocol on the physical layer.

To establish k , we use the DHKE. The prover chooses $\bar{x} \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\bar{q}}$, the verifier chooses $\bar{y} \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\bar{q}}$ and they exchange $\bar{X} = \bar{g}^{\bar{x}}, \bar{Y} = \bar{g}^{\bar{y}}$ and both compute $k = \bar{g}^{\bar{x}\bar{y}}$, for some generator \bar{g} in a Diffie-Hellman (DH) [24] group of prime order \bar{q} . The prover also computes $\hat{X} \stackrel{\mathcal{C}}{\leftarrow} \text{VRF}_{sk_P}(\bar{X}) = \hat{\mathbf{g}}^{1/(sk+\bar{X})}$ and sends to the verifier. They use the agreed upon key $k = \bar{g}^{\bar{x}\bar{y}}$ as the key for UWBPR, instead of a static k .

Signature Reblinding The prover blinds the signature σ : Choose $r_1, r_2 \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\hat{q}}$ and compute $\tilde{\sigma} = (\tilde{a}, \tilde{A}, \tilde{b}, \tilde{B}, \tilde{c})$, where

$$\tilde{a} = a^{r_1}, \quad \tilde{A} = A^{r_1}, \quad \tilde{b} = b^{r_1}, \quad \tilde{B} = B^{r_1}, \quad \tilde{c} = (c^{r_1})^{r_2}.$$

The prover sends $\tilde{\sigma}$ to the verifier. Now, the prover and verifier each compute

$$v_x \leftarrow e(\hat{X}, \tilde{a}), \quad v_{xy} \leftarrow e(\hat{X}, \tilde{b}), \quad V_{xy} \leftarrow e(\hat{X}, \tilde{B}), \quad v_s \leftarrow e(\hat{g}, \tilde{c}).$$

This blinding procedure can be done once, there is no point in presenting the verifier with several different blindings of the same signature.

Setup. Now we prepare for the actual distance bounding. The prover chooses

$$\rho \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\hat{q}}, \quad \bar{\rho} \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\hat{q}}, \quad \hat{\rho}_{sk} \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\hat{q}}, \quad \hat{\rho}_r \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\hat{q}}, \quad \hat{\rho}_{r'} \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_{\hat{q}},$$

computes

$$R \leftarrow g^\rho, \quad \bar{R} \leftarrow \bar{g}^{\bar{\rho}}, \quad \hat{R} \leftarrow \text{VRF}_{sk}(\bar{X})^{\hat{\rho}_{sk}}, \quad \hat{R} \leftarrow v_s^{\hat{\rho}_{r'}} v_{xy}^{\hat{\rho}_{sk}} V_{xy}^{\hat{\rho}_r}$$

and sends R, \bar{R}, \hat{R} to the verifier.

The verifier chooses m challenges of length l ,

$$c_1 \xleftarrow{\$} \{0, 1\}^l, \dots, c_m \xleftarrow{\$} \{0, 1\}^l,$$

and sends these to the prover. The prover computes responses

$$s_i \leftarrow \rho - c_i \alpha, \quad \bar{s}_i \leftarrow \bar{\rho} - c_i \bar{\alpha}, \\ \hat{s}_i^{(r')} \leftarrow \hat{\rho}_{r'} + c_i r', \quad \hat{s}_i^{(sk)} \leftarrow \hat{\rho}_{sk} - c_i sk, \quad \hat{s}_i^{(r)} \leftarrow \hat{\rho}_r - c_i r$$

for $i \in \{1, \dots, m\}$, where $r' = r_2^{-1}$.

Distance Bounding. The verifier decides which of the m challenges to use, $i \xleftarrow{\$} \{1, \dots, m\}$, sends its decision i to the prover and starts its clock. The prover instantly replies with the pre-computed $s_i, \bar{s}_i, \hat{s}_i^{(r')}, \hat{s}_i^{(sk)}, \hat{s}_i^{(r)}$. Note that these values must be sent as a concatenation in *one* UWBPR reply. The verifier stops the clock and records the round-trip time Δt .

Verification. The verifier checks that

$$R = A^{c_i} g^{s_i}, \quad \bar{R} = \bar{X}^{c_i} g^{\bar{s}_i}, \quad \hat{R} = \left(\hat{\mathbf{g}} \hat{X} \bar{X} \right)^c \hat{X}^{\hat{s}_i^{(sk)}}, \quad v_x^{c_i} \hat{R} = v_s^{s_{r'}} v_{xy}^{s_{skP}} V_{xy}^{s_r}$$

and that the round-trip time $\Delta t < t_{\max}$ does not exceed the time-limit t_{\max} (corresponding to the desired distance bound). Finally, the verifier checks the validity of $\hat{\sigma}$:

$$e(\tilde{a}, \hat{Z}) = e(\hat{g}, \tilde{A}), \quad e(\tilde{a}, \hat{Y}) = e(\hat{g}, \tilde{b}), \quad e(\tilde{A}, \hat{Y}) = e(\hat{g}, \tilde{B}).$$

The setup, distance bounding and verification phases are repeated n times.

4.1 Security Analysis

There are two questions we must answer about this protocol:

- (1) Can the adversary perform a MITM attack against the DHKE?
- (2) Does \hat{X} break anonymity?

To justify the first question, we use a plain DHKE to agree on k , start using k for DB and during the DB do the authentication. To justify the second question, we use CL04 anonymous credentials [17]. These provide anonymity. However, revealing $\hat{X} = \text{VRF}_{sk}(\bar{X})$ is not part of CL04. So we must show that \hat{X} maintains anonymity.

To answer the first question, by Camenisch [16], the above protocol is a ZKPK. In the case of a MITM, the prover will run the protocol using \bar{X} and sk ,

while the verifier will use \bar{X}' , possibly modified by the adversary. The adversary's goal is to pass as knowing sk and its signature (to impersonate real hardware). The sk must be the same sk throughout the subprotocols [16]. In one subprotocol, the prover will prove knowledge of $\text{VRF}_{sk}(\bar{X})$. If $\bar{X}' \neq \bar{X}$ used by the verifier is wrong, then by soundness the proof will fail.

To answer the second question, $\hat{X} = \text{VRF}_{sk}(\bar{X})$ is the output of the VRF of Dodis and Yampolskiy [25] keyed by sk . This is a pseudo-random function (PRF) with a proof of correctness. If an adversary can use \hat{X} to tell one device from another, then the adversary can distinguish VRF from a random function [25].

5 Bit-by-Bit Distance-Bounding Schnorr Protocol

We will now continue with a more traditional (in DB terms) version of the protocol, one that uses bit-by-bit communication.

We present the protocol in Fig. 4. We will present the protocol as a DB version of the Schnorr identification scheme ($\text{PK}\{(\alpha) : A = g^\alpha\}$), this is for the sake of exposition. It can be generalized to e.g., CL04 [17].

The (cyclic) group with generator g with prime order q is a system parameter. The private key α , with public key $A = g^\alpha$, is generated once by the prover in the setup phase. The verifier has a copy of the public key A and wants to verify that the private key α is within a certain distance-bound.

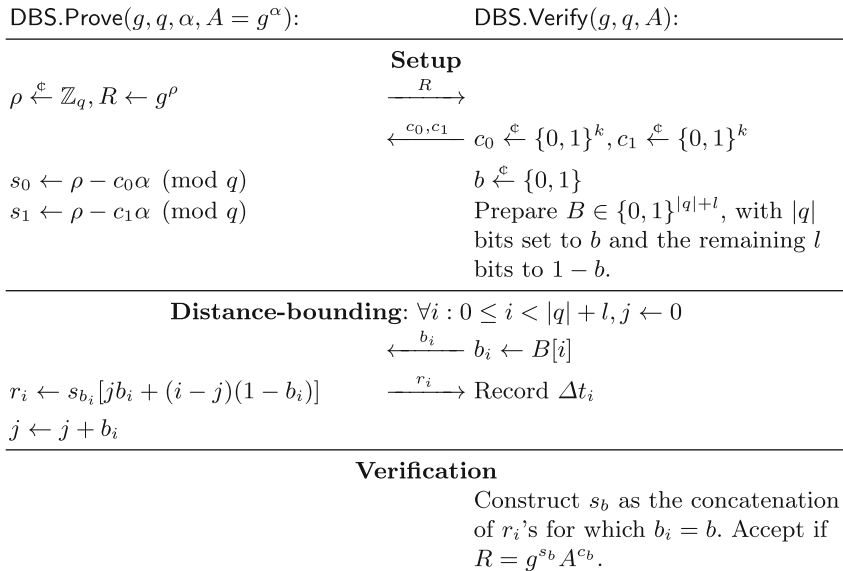


Fig. 4. One-round protocol instance of the DBS.Prove \leftrightarrow DBS.Verify DB Schnorr protocol for $\text{PK}\{(\alpha) : A = g^\alpha\}$. The protocol should be repeated n times to achieve the desired soundness and distance-bounding errors. We let $s_{b_i}[n]$ denote the n th bit of s_{b_i} .

During one round, the prover commits to a random nonce: more precisely he chooses $\rho \xleftarrow{\mathcal{C}} \mathbb{Z}_q$ uniformly at random, computes $R \leftarrow g^\rho$ and sends R to the verifier. The verifier generates two challenges $c_0 \xleftarrow{\mathcal{C}} \{0, 1\}^k, c_1 \xleftarrow{\mathcal{C}} \{0, 1\}^k$ and sends them to the prover. The prover computes $s_0 \leftarrow \rho - c_0\alpha, s_1 \leftarrow \rho - c_1\alpha$. This step is the main difference to the version in Sect. 4: since we use bit-sized challenges, we can only choose between two challenges (instead of m).

We let $|q|$ denote the length of q in bits, so any element $z \in \mathbb{Z}_q$ can be represented by $|q|$ bits. The verifier will request all $|q|$ response-bits from one challenge (say s_b) and only $0 < l \leq |q|$ from the other (s_{1-b}). Then the verifier can authenticate the prover by checking if $R = g^{s_b} A^{c_b}$.

This must be repeated for n rounds.

5.1 Security Analysis

We can prove this protocol secure against MF, DH and DF; but not against TF. Those proofs are available in the full version of the paper. Here, we will discuss only the most crucial differences.

The l -bit Problem. We can show that the system of linear equations created by the protocol and consisting of leaked bits from the second reply is under-determined. However, we cannot get information theoretic security, that would require leaking no bits, but we leak some bits. The only lead an adversary get to those remaining unknown bits are the commitments. The adversary has a commitment to the randomness r_i in the form of $R_i = g^{r_i}$ and a commitment to α in the form of $A = g^\alpha$. This means that

$$\begin{aligned} R &= A^{-c'} g^{s'} = A^{-c'} g^{s'[0]2^0 + \dots + s'[m-1]2^{m-1}} \\ &= A^{-c'} g^{s'_{\text{known}}} g^{s'_{\text{unknown}}}. \end{aligned}$$

We will now show how the hardness of finding s'_{unknown} relates to the hardness of finding discrete logarithms.

Definition 7 (Discrete Logarithm Problem). *Let g be a generator for a group of prime order q . Given $g, q, A = g^\alpha$, compute α .*

Definition 8 (DB-Schnorr Problem). *Let g be a generator for a group of prime order q . Give $g, q, R = g^\rho A = g^\alpha$ to the adversary. Let the adversary choose c and additionally give him \hat{s} such that \hat{s} are the l least significant bits of $s = \rho + c\alpha \pmod{q}$. The adversary wins if he returns \hat{s}' such that $\hat{s} + \hat{s}' = s \pmod{q}$.*

Lemma 1. *Given an algorithm A_{DBS} that solves the DB-Schnorr Problem (Definition 8) with probability ϵ , we can construct an algorithm A_{DL} that solves the Discrete Logarithm Problem (DLP) (Definition 7) with probability $\epsilon 2^{-(l+k)}$.*

The proof of this lemma can be found in the full version of the paper.

Zero-Knowledge and Proof of Knowledge. The main difference between this protocol and the Schnorr protocol is that we have the prover compute responses for two different challenges, c_0, c_1 . In the authentication step, the verifier chooses only one of those challenges. However, the verifier requests all $|q|$ bits of s_b and l bits of s_{1-b} .

This is a proof of knowledge: Completeness and soundness follows from the Schnorr protocol. The knowledge extractor works exactly the same. It does not matter that the verifier first chooses two challenges and later reveals which one to use and the prover sends the response bit-by-bit.

The change is with the zero-knowledge property. This is *computational* zero-knowledge: The simulator chooses

$$s_0, s_1 \stackrel{\mathcal{C}}{\leftarrow} \mathbb{Z}_q; \quad c_0, c_1 \stackrel{\mathcal{C}}{\leftarrow} \{0, 1\}^k; \quad b \stackrel{\mathcal{C}}{\leftarrow} \{0, 1\}.$$

Then it can set

$$R_0 \leftarrow g^{s_0} A^{c_0}, \quad R_1 \leftarrow g^{s_1} A^{c_1}.$$

Now the simulator must come up with those l bits. By Lemma 1 and some lemmas in the full version of the paper, we can see that distinguishing these bits from random is comparable to solving the DLP. So, the simulator can simply choose the l bits $L \stackrel{\mathcal{C}}{\leftarrow} \{0, 1\}^l$ uniformly at random.

The simulator outputs R_b, c_0, c_1, s_b, L .

Why Not TF? The problem is that asymptotically, l and $l + 1$ are rather close. So the colluding parties can exchange $l + 1$ bits instead of just l . This way they can create a buffer of responses. However, this means that they can stretch the distance bound; at some point, $l + \epsilon$ will be too large and make α easy enough to guess. So the incentive changes from the legitimate party not wanting to cheat, into the legitimate party not wanting to cheat “too much”. However, we have not formally treated this part more than concluding that we cannot prove TF resistance in terms of Definitions 3 and 4.

6 Performance

We will now review the performance of the proposed protocols. Since the construction is designed as a drop-in replacement for the Schnorr identification-scheme [10] as a ZKPK protocol, we will focus on how much we must “pay” for the DB property. We summarize the results in Table 1.

Mutual Trust. We start with the mutual-trust version of the protocol from Sect. 3.1 (see Fig. 2).

In *one round* of the protocol, the cost for the verifier is that of generating m number of l -bit strings, instead of only one. This costs the verifier a factor m of randomness. The cost for the prover is that of computing m number of replies ($s \leftarrow \rho - c\alpha \pmod{q}$), instead of only one.

To achieve security, the protocol must be repeated n times. We have the l -parameter which controls the soundness of the ZKPK and the m -parameter

Table 1. A summary of how costly distance bounding is in terms of randomness, arithmetic operations (addition, multiplication), exponentiations, and pairing operations. $m, l \in \mathcal{O}(\log \lambda)$ and we must repeat the protocol n times to achieve desired soundness.

Protocol	Rand. (bit)	Arith. (+, \times)	Exp.	Pairings
Schnorr	$(\lambda + l)n$	$3n$	$3n$	0
DB, mutual trust	$(\lambda + ml)n$	$3mn$	$3n$	0
DB, PKI (total)	$4\lambda + (5\lambda + ml)n$	$(10m + 8)n$	$9 + 16n$	14
—DHKE	2λ	0	4	0
—CL04 blind	2λ	0	5	8
—DB	$5\lambda + ml$	$10m + 2$	6	0
—Verification	0	6	10	6

controls the soundness of the distance bounding. If we want to achieve 80 bits of security, we can let $l = m = 6$ (remember, l must be logarithmic in the security parameter) and $n = 14$. This makes distance bounding a factor 6 more expensive over the *malicious-verifier secure* Schnorr protocol.

PKI. The PKI version of the protocol (Sect. 4 and Fig. 3) performs one signature verification and four additional proofs in parallel. The cost for the verifier is one signature verification (CL04 [17]) in addition to the factor m of randomness used. On the prover side, we have the computations for the signature verification and four additional proofs in parallel, this changes the cost to a factor of $5m$ for the prover: the prover must compute $s_i = \rho - c_j \alpha_i$ for $i \in \{1, \dots, 5\}, j \in \{1, \dots, m\}$.

The cost of the signature is the same as for Anon-Pass [5], which implements a public transport pass using this signature scheme. This signature verification introduces more additions, multiplications, exponentiations and pairing operations. However, the signature will only be blinded and verified once, so this represents a constant factor. It is sufficient to verify it once and reuse the same blinding for all n repetitions of the protocol. (The same is true for the DHKE.) Thus it is only the proof of knowledge that must be rerun n times.

Bit-by-Bit The traditional bit-by-bit version of the protocol (Sect. 5, Fig. 4) is not as efficient. That is due to $m = 2$ being fixed in this case (there are only two responses to choose from), and consequently, we must have a larger value for n to achieve the desired security.

7 Conclusion

In this paper, we have adapted the Schnorr identification scheme [10] to work as DB ZKPK for discrete logarithms. It allows us to construct DB, general attribute-based authentication; with more general attributes than provided by

previous DB protocols. We constructed, proved, and formally verified three versions of the protocol: one that is more efficient and resistant to the attacks leveled at DB protocols, but that assumes that more than one bit at a time can be sent in the fast phase (as demonstrated by Singh, Leu, and Capkun [14]). For another, we developed a PKI scheme to generalize [14] to work for scenarios such as ours, where the prover and verifier do not trust, or even know, each other. The third version is a classic bit-by-bit protocol that is not fully TF resistant, but otherwise fulfills all requirements.



References

1. Brands, S., Chaum, D.: Distance-bounding protocols. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 344–359. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48285-7_30
2. Desmedt, Y., Goutier, C., Bengio, S.: Special uses and abuses of the Fiat-Shamir passport protocol (extended abstract). In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 21–39. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_3
3. Desmedt, Y.: Major security problems with the ‘unforgeable’(Feige)-Fiat-Shamir proofs of identity and how to overcome them. Proc. SECURICOM **88**, 15–17 (1988)
4. Cremers, C., Rasmussen, K.B., Schmidt, B., Capkun, S.: Distance hijacking attacks on distance bounding protocols. In: 2012 IEEE Symposium on Security and Privacy (SP), pp. 113–127. IEEE (2012)
5. Lee, M.Z., Dunn, A.M., Waters, B., Witchel, E., Katz, J.: Anon-pass: practical anonymous subscriptions. In: 2013 IEEE Symposium on Security and Privacy (SP), pp. 319–333. IEEE (2013)
6. Mauw, S., Smith, Z., Toro-Pozo, J., Trujillo-Rasua, R.: Distance-bounding protocols: verification without time and location. In: IEEE S&P 2018 (Oakland), pp. 152–169 (2018). <https://doi.org/10.1109/SP.2018.00001>
7. Bussard, L., Bagga, W.: Distance-bounding proof of knowledge to avoid real-time attacks. In: Sasaki, R., Qing, S., Okamoto, E., Yoshiura, H. (eds.) SEC 2005. IAICT, vol. 181, pp. 223–238. Springer, Boston, MA (2005). https://doi.org/10.1007/0-387-25660-1_15
8. Bay, A., Boureau, I., Mitrokotsa, A., Spulber, I., Vaudenay, S.: The Bussard-Bagga and other distance-bounding protocols under attacks. In: Kutyłowski, M., Yung, M. (eds.) Inscrypt 2012. LNCS, vol. 7763, pp. 371–391. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38519-3_23
9. Vaudenay, S.: Sound proof of proximity of knowledge. In: Au, M.-H., Miyaji, A. (eds.) ProvSec 2015. LNCS, vol. 9451, pp. 105–126. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26059-4_6
10. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991). <https://doi.org/10.1007/BF00196725>
11. Dürholz, U., Fischlin, M., Kasper, M., Onete, C.: A formal approach to distance-bounding RFID protocols. In: Lai, X., Zhou, J., Li, H. (eds.) ISC 2011. LNCS, vol. 7001, pp. 47–62. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24861-0_4
12. Fischlin, M., Onete, C.: Terrorism in distance bounding: modeling terrorist-fraud resistance. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 414–431. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38980-1_26

13. Avoine, G., et al.: A terrorist-fraud resistant and extractor-free anonymous distance-bounding protocol. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ASIA CCS 2017, pp. 800–814 (2017). <https://doi.org/10.1145/3052973.3053000>
14. Singh, M., Leu, P., Capkun, S.: UWB with pulse reordering: securing ranging against relay and physical-layer attacks. In: 26th Annual Network and Distributed System Security Symposium, NDSS 2019 (2019)
15. Diffie, W., Van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. *Designs Codes Cryptogr.* **2**(2), 107–125 (1992). <https://doi.org/10.1007/BF00124891>
16. Camenisch, J.: Group signature schemes and payment systems based on the discrete logarithm problem. Ph.D. thesis, ETH Zurich, Zürich, Switzerland (1998). <http://d-nb.info/953066045>
17. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_4
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. Boureanu, I., Mitrokotsa, A., Vaudenay, S.: Practical and provably secure distance-bounding. *J. Comput. Secur.* **23**(2), 229–257 (2015)
20. Damgård, I.: On Σ -protocols. In: CPT 2010, vol. 2. <http://www.cs.au.dk/~ivan/Sigma.pdf>
21. Gambs, S., Lassance, C.E.R.K., Onete, C.: The not-so-distant future: distance-bounding protocols on smartphones. In: Homma, N., Medwed, M. (eds.) CARDIS 2015. LNCS, vol. 9514, pp. 209–224. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31271-2_13
22. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Persiano, G., Galdi, C. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36413-7_20
23. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, pp. 132–145 (2004)
24. Diffie, W., Hellman, M.: New directions in cryptography. *IEEE Trans. Inf. Theory* **22**(6), 644–654 (1976)
25. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30580-4_28



Trenchcoat: Human-Computable Hashing Algorithms for Password Generation

Ruthu Hulikal Rooparaghunath^(✉) , T. S. Harikrishnan,
and Debayan Gupta 

Ashoka University, Sonipat, Haryana, India
{ruthu.rooparaghunath_ug21, debayan.gupta}@ashoka.edu.in,
ts.harikrishnan@alumni.ashoka.edu.in

Abstract. The average user has between 90–130 online accounts [17], and around 3×10^{11} passwords are in use this year [10]. Most people are terrible at remembering “random” passwords, so they reuse or create similar passwords using a combination of predictable words, numbers, and symbols [16]. Previous password-generation or management protocols have imposed so large a cognitive load that users have abandoned them in favor of insecure yet simpler methods (*e.g.*, writing them down or reusing minor variants).

We describe a range of candidate *human-computable* “hash” functions suitable for use as password generators - as long as the human (with minimal education assumptions) keeps a single, easily-memorizable ‘master’ secret - and rate them by various metrics, including *effective security*. These functions hash master-secrets with user accounts to produce sub-secrets that can be used as passwords; $F_R(s, w) \rightarrow y$, which takes a website w and produces a password y , parameterized by the master secret s , which may or may not be a string.

We exploit the unique configuration R of each user’s associative and implicit memory (detailed in Sect. 2) to ensure that sources of randomness unique to each user are present in each F . An adversary cannot compute or verify F_R efficiently since R is unique to each individual; in that sense, our hash function is similar to a physically unclonable function [37]. For the algorithms we propose, the user need only complete primitive operations such as addition, spatial navigation or searching. *Critically, most of our methods are also accessible to neurodiverse, or cognitively or physically differently-abled persons.*

Given the nature of these functions, it is not possible to directly use traditional cryptographic methods for analysis; so, we use an array of approaches, mainly related to entropy, to illustrate and analyze the same. We draw on cognitive, neuroscientific, and cryptographic research to use these functions as improved password management and creation systems, and present results from a survey ($n = 134$ individuals, with each candidate performing 2 schemes) investigating real-world usage of these methods and how people *currently* come up with their passwords. We also survey 400 websites to collate current password advice.

Keywords: Usable security · Applied cryptography · Hash functions · Security policy · Authentication · Identification

1 Introduction

Your password must be between 8–16 characters long, with at least one uppercase character, one lowercase character, one number, and one special character (such as !, @, #, etc.), must not include your username, and be changed every 90 days.

Memorizing myriad passwords, with (often questionable) constraints imposed to make each password as “random” as possible, and little guidance on how to manage this information, is a herculean task. This has resulted in people using easily guessable and common passwords [30]. Surveys last year indicated that individuals reuse over half of all passwords for multiple accounts, with many others being easily attacked with a dictionary of common passwords [16].

Anecdotally, users prioritize convenience over privacy when accessing newsletters, spam mails, or magazine subscriptions. They assign important accounts with less conveniently memorable passwords. This trade-off in memorability results in compromised security when passwords are written and stored at home. [34] Weak passwords are a serious threat when they guard sensitive data or systems, and may lead to identity theft, insurance fraud, public humiliation, etc. [43].

Common approaches to handling this rely on instructing users to create ‘strong’ passwords with suggestions such as: ‘don’t use your name or birth-date’, ‘include symbols’ and ‘don’t capitalize only the first letter’. However, users routinely ignore or circumvent these suggestions because of their cognitive load.

The current standard for password management and security is a password manager. Unfortunately, several sources report serious flaws (including zero-day attacks) *consistently* found in the most popular password managers every year [3, 15]. Some managers are also vulnerable because of their tendency to store the passwords to the password manager in plaintext.¹

Digital and physical copies of passwords will always have vulnerabilities, but remembering several passwords imposes a cognitive load that users are unwilling or unable to manage. Past research has proposed several password generation methods [4–6] but those that consider real-world usage have not been tested beyond a dozen people [4], or have placed too large a cognitive load on users.

We propose a family of **public** derivation functions F such that, if we start with a master secret, s (which the human memorizes), we can derive a sub-secret y_i for each website w_i . Broadly, our requirements for such F would be: (1) Given (y_i, w_i) , where $y_i = F_R(s, w_i)$, it should be computationally hard to find s ; (2) Given $(y_1, w_1), (y_2, w_2), \dots, (y_k, w_k)$ and w_{k+1} , it should be computationally hard to find y_{k+1} (secure as in Unforgeability under Random Challenge Attack [6]). This minimizes cognitive load by requiring only the memorization

¹ Preventing this, in most password managers, requires users to terminate the manager each time after use. Users may be unaware of this or disregard it because of inconvenience, which once again lowers its security [25].

of s , with any y_i being derived using public w_i and F . *Critically, s , unlike y_i , need not be a string!* (We discuss visual and cue-based s in Sect. 2.)

F must be easily human-computable. F must also not require too much aid, to minimize cognitive load. Further, for individuals to reproduce the same password each time, F should be deterministic with respect to each individual. One way to satisfy most of these requirements is through a cryptographically secure hash.

Predefined cryptographic hash functions such as SHA-3 (with preset size parameters, and conversion to appropriate characters) could be used in place of F , calculating $y = F(s \cdot w)$, concatenating s and w where s is a string. Unfortunately, most humans cannot easily compute SHA-3 in their heads. We need something that includes *some* features of a cryptographically-secure hash function without requiring the mathematical heavy-lifting common to such schemes. In the rest of this document, we describe a number of approaches to finding the same, and the results of our survey on the subject. (Assumptions made by cryptographers on what laypersons would find “easy to compute” may be incorrect; we must empirically observe the methods people are willing and able to use.)

1.1 Paper Outline and Contributions

To optimize our hash functions for human use, we discuss visual cues, and implicit and associative methods suggested by cognitive and neuro-scientific research in Sect. 2. Previous literature on human-computable passwords requires rehearsal schedules, online aid, etc. with various caveats and problems [4–6].

These issues are obviated by using an easily-memorized key with human-computable algorithms designed for password generation and management. Section 4.1 presents a range of such hashing algorithms. An adversary cannot compute or verify these hashes efficiently, since these are unique to each individual; in that sense, our hash function is similar to a physically unclonable function [37]

In this context, we discuss *effective security* in Sect. 5.2 which weighs cryptographically evaluated security against human usability. *E.g.*, generating random passwords without associative memory techniques or computational tools and writing materials may impose large cognitive loads, reducing usability²

We also define *graceful degradation* - our algorithms retain a significant amount of their effective security even if access to writing materials, computers, or the internet is unavailable. We test the algorithms presented in this paper as well as Cue Pin Select [4] on a survey population of 134 individuals (with each person assigned to two, randomly-chosen schemes), averaging 56 responses per algorithm from people between the ages of 18 to 25. We analyse the results in Sect. 5.1 and also use an LSTM to test character predictability in Sect. 5.4.

We cannot use standard cryptographic techniques to evaluate our schemes, as they are explicitly optimized for representation in human brains but difficult to

² In general, as a human-computable hash function grows in difficulty, a human is more likely to abandon it [16,30] and revert to weak password practices. So, one can have very high theoretical security but, in practice, be totally insecure.

represent or simulate on computers (thus contributing to their security). So, we introduce metrics to assess the security of human-computable schemes, measure ease of use, rememberability, unforgeability under Random Challenge Attack [6], and more, in Appendix A. We also classify algorithms based on their paradigms, limiting factors, and success of password recall in Sect. 5.2.

Section 3 discusses common password hygiene errors and current password advice; we survey 400 websites and applications for such advice (Table 1). We also provide insight into real-world methods individuals *currently* use to come up with passwords in Sect. 6. Finally, Sect. 5 uses our survey results to understand the determinism or stability of our schemes during real-world usage.

2 Cognitive and Neuro-Scientific Perspectives

During WW1, before the advent of powerful computers, soldiers used “trench codes” to communicate across trenches. These had to be designed to be computable by soldiers under pressure without assuming high education levels – this involved coming up with clever codebooks/manuals³. Such trench codes had their own problems, of course, but these issues were obviated by the time WW2 came around; ever since then, we have optimized our cryptographic functions (encryption, hashing, etc.) for increasingly-powerful computers, not humans. To design human-computable functions while maintaining security, we must first discuss how to optimize functions for the human brain.

Broadly, the brain manages memory in two categories [5]: persistent (e.g., notepads) and associative (human memory). The latter is clearly more secure for password storage and recollection, as elaborated in the Introduction. Password recollection depends on the conscious retrieval of detailed memory, which imposes a large cognitive load (so users create workarounds to ease this load). Relying on visual, implicit and associative memory can ease this cognitive load.

Visual memory is capable of long-term storage of large amounts of detailed information. Implicit, associative memory aids in lasting rapid recall. However memorizing large amounts of new visual information requires constant rehearsal to become embedded in memory, which is tedious. Fortunately, humans already accumulate a vast amount of long-term information throughout their lives. Sub-conscious rehearsal repeated over time does *not* feel tedious: drawing on implicit memory - such as repeatedly navigating a house - requires less effort.

Visually cued recollection is easier than explicit recollection [2]. This is also a more accessible method, as neurologically damaged or disabled patients can succeed at implicit memory tasks, even when they cannot succeed on explicit memory tasks [31]. We thus contend that password retention relying on implicit memory retrieval has the potential to be *stable, long-lasting, and equitable*.

Some functions proposed in Sect. 4.1 are based on this capacity for detailed storage and fast retrieval in visual memory. The *Memory Palace* method uses

³ Beyond careful design, these also included side-channel defenses e.g., the paper material was designed to degrade within a few weeks, ensuring that obsolete codes would not be used, and “lost” manuals would lose value quickly.

visually-cued subkey recollection. This can be further improved by using physical copies of partial visual images for cues, eliminating the cognitive load of remembering visual cues themselves. (See Sect. 4 for details of these protocols.)

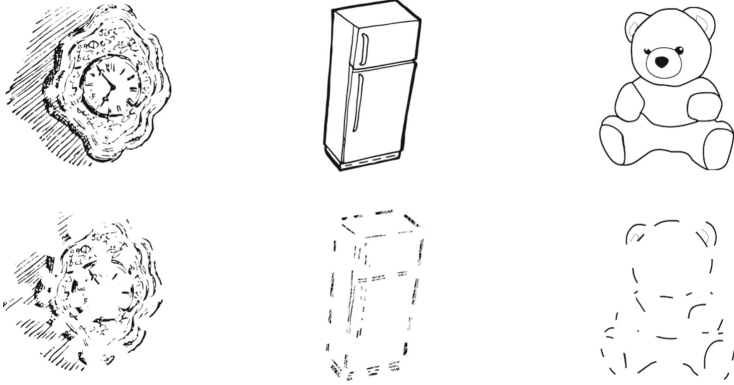


Fig. 1. Complete and partially complete line drawings for visually-cued subkey priming based on user subkeys from the Memory Palace.

We now briefly explore the act of using partial images as visual cues (Fig. 1) for password-subkey retrieval. We define p_i as the probability with which a random user correctly identifies a partial image such as above, when they are primed on the original completed image i . n_i is the probability that they identify the partial image if they are not primed on the original image.

The priming effect⁴ is α , with $\alpha > 0$ and $p_i \geq n_i + \alpha$ i.e. the probability of correctly identifying partial images with priming is greater than the probability of correct identification without priming [11]. Users may choose to use cues for all of their accounts, which would have required 130 cue-subkey associations for the average user last year. [17] However, this is unlikely and most users may deploy hash functions and cues for only the most sensitive data.

What remains then, is to evaluate the success of an adversarial (without cue-subkey associations) attack. Fortunately, this is well-established in neuroscientific literature; we paraphrase [11]: *Assuming an adversary knows p_i, n_i , and the correct label for that image, an optimal adversarial strategy is to maximize the probability of recovery of those images without knowledge of the set U on which the user was primed (since this set U exists uniquely in the mind of each user). The best strategy is to label each image correctly at random. However supposing an adversary is allowed to recover a user’s [password] with probability at most 0.5% (false positive rate). For valid recovery to succeed at least 97.5% of*

⁴ All images have demonstrably high priming “strength” [31] i.e. our images are already embedded in the user’s mind (familiar places that they can navigate mentally).

the time (false negative rate of 2.5%), a user would need to correctly label 135 images without prior knowledge to recover a word.⁵

Users surveyed during a 2004 survey on Password Memorability and Security [42] were observed to use their own password generation methods, which were usually weak, yet met the security requirements demanded by websites. We thus propose that exploiting users’ unique configurations of memory as a source of randomness enables compelling, secure password generation.

3 Password Security Advice

There are three common password-hygiene errors [40] – choosing simple passwords (123456, *iloveyou*, *qwerty*, etc.), insecure storage, and password reuse. Attacks⁶ include guessing (common passwords), brute force, and dictionary attacks. The passwords mentioned above have 28, 40, and 32 bits of entropy respectively, which require around half a million attempts [12] to crack. (In reality, a hacker would guess common passwords first, and thus break these easily.) With the aid of GPU supported tools like Hashcat, Rainbow Crack etc., a 9-character password can be cracked in an alarmingly short time [41] – around 18 min to check salted hashes for every 9-character password, assuming ideal conditions⁷.

Given these issues, many websites/applications suggest strategies users should follow to create secure passwords. To better understand such password advice, we surveyed 400 highly visited platforms, compiled manually and through public lists [1, 19, 36, 39]. Of these, 54 offered password advice; see Table 1 for a summary.

Table 1. Advice from 400 highly-visited websites and apps (54 provided advice).

Summary of Password Advice	
<i>Parameters Suggested</i>	<i>% of platforms</i>
Length (<6 characters)	20%
Length (≥ 6 characters)	20%
Length (≥ 8 characters)	41%
Length (≥ 10 characters)	19%
Numerals	83%
Uppercase	65%
Special Characters	63%
Password Managers	9%

⁵ See [11] for a detailed proof.

⁶ Cracking means an adversary with access to password hashes, has found a collision.

⁷ In practice, the time taken to find a password’s hash depends on the alphabet used, degree of parallelization, hardware specifications such as processor flops, etc. [8].

Websites suggest tactics such as intentionally misspelling words, replacing letters (‘@’ for ‘a’, ‘\$’ for ‘s’, etc., so that ‘its raining cats and dogs’ become ‘1tsrAIn1NGc&t&DGS!’). However, there exist various dictionaries of special characters, common misspellings, and symbol substitutions. Hence, such tricks are ineffective against modern hackers [32]. An attack on with these dictionaries exposed hash collisions such as “Apr!221973,” and “Qbesancon321”.

What, then, is a secure password? The RSA challenge by *RSA Laboratories* [38] issued random keys from 40 upto 128 bits with ciphertexts. Distributed.net has been working on the 72 bit key for over 6400 days as of July, 2020 [35]; at this pace, it takes around 200,000 days to search the entire keyspace. Currently, 72 bits of entropy provide sufficient security; 80 bits of entropy are recommended for long-term security [38].

4 Human-Computable Hashing Algorithms

The functions proposed here draw upon the ideas discussed in Sect. 2 to balance security and ease of use. We describe all algorithms and provide examples for cases that might otherwise be confusing. Algorithms were primarily designed to determine which approaches (subkey-generation, visualization, addition, implicit association etc.) produce the most effective and secure passwords. For this reason, they vary widely and cover a range of password generation tactics.

We perform a naive entropy calculation (assuming letter entropy values are independent) for the purposes of comparing hashing algorithms. *These numbers should **not** be taken seriously as proxies for security in and of themselves, but may be useful for comparison.* Difficult-to-use schemes might push users to simply write the password down (or ignore the scheme). A “good” function produces high entropy passwords that are easy to compute.

Typically, hash function security is judged by pre-image resistance, collision resistance, randomness, etc. [14]. That is not easily done for our functions – we cannot generate billions (or even millions) of hashes, as the process of generation relies on individuals’ unique memory representations and sources of randomness (discussed below and in Appendix A). We discuss some metrics we can use in Sect. 5 and cryptographic details in Appendix A.

4.1 Description of the Schemes

We describe the following human-computable hash functions: Memory Palace, Scrambled Box, Song Password, Internal Sentence. w is the website name, s is the single secret user key, and h is the candidate for F . F and h are functions of R , the unique configuration of each user’s memory. Each source of randomness is indicated by R and specified at the end of each algorithm. Sources are elaborated on in Appendix A. Common sources of randomness across all algorithms: unique memory associations; choosing between symbols, numerals or letters on the same key.

Memory Palace. s : A location_R very familiar to the user. $h_R(s, w)$:

- Step 1 *subkey generation* Mentally navigate the location using each letter in w . For vowels turn left and walk straight_R, else turn right and walk straight. After reaching the end of the website name, think of a word (or words) that describe what the user faces. (If $w = gmail$, visualizing a familiar location, mentally move right and straight twice then left and straight twice, then right and straight twice. $s =$ a description of what you face.)
- Step 2 *group sum* Divide the word(s) into groups of 2 letters (pairs). Sum each group using letter values to create a new letter. (Letters map to $\{a = 1, \dots, z = 26\}$, if sum overflows, subtract 26 from the sum.) If s can't be evenly split add a favorite letter_R to the end. (If $s =$ white birds, split into wh, it, eb, ir, ds. Sum into $w + h = e, i + t = c, e + b = g, i + r = a, d + s = w$)
- Step 3 *group character* If the first letter of a pair is a vowel, write the symbol/letter above and to its immediate diagonal_R left on the keyboard after the letter from the group sum. Else, the symbol/letter above to its immediate diagonal right on the keyboard. (Described and illustrated visually during the survey.) *password*: Alternate group sum and group character. (Alternating group sum letters with corresponding diagonal symbols, *password* = e3cfgya1w3.)

Randomness: Spatial characteristics of direction, number of steps to take when walking. Letter preference when appending letters to make the length of s even. Interpretation of diagonal angle, choosing the i^{th} symbol along the diagonal.

Scrambled Box. *global*: A 10×10 table of symbols, numbers and letters (repetitions allowed). Movements associated with each story element (can be changed): Sad = up; Memorable characters (Animals, Villains etc.) = diagonal to the right and down; Events that move the story forward = horizontal to the right; Happy = move to the opposite corner of the table (Fig. 2).

	0	1	2	3	4	5	6	7	8	9
0	A	K	U	4	!	_	!	s	\	*
1	B	L	V	5	@	+	j	t	:	<
2	C	M	W	6	#	a	k	u	'	>
3	D	N	X	7	\$	b	l	v	.	?
4	E	O	Y	8	%	c	m	w	.	A
5	F	P	Z	9	^	d	n	x	/	a
6	G	Q	'	0	&	e	o	y	{	B
7	H	R	1	.	*	f	p	z	}	b
8	I	S	2	=	(g	q	[]	C
9	J	T	3	~)	h	r]	:	c

	0	1	2	3	4	5	6	7	8	9
0	7	!	_	9	^	d	G	Q	'	t
1	X	@	+	0	&	e	H	R	1	K
2	C	M	A	.	*	f	I	S	2	=
3	D	N	B	L	\$	b	J	T	3	~
4	E	O	Y	8	%	c	m	w	.	A
5	F	P	Z	s	6	i	n	x	/	a
6	U	4	\	*	W	j	o	y	{	B
7	V	5	:	<	#	a	p	z	}	b
8	k	u	'	>	(g	q	[]	C
9	l	v	.	?)	h	r]	:	c

Fig. 2. Example 10×10 box and S-box, with scrambling highlighted

s : A well-known easily remembered story_R name. $h_R(s, w)$:

- Step 1 *S-box generation*. Find 4 elements (e.g., emotions, events, memorable characters) in the story’s plot and write them down in order. For the x^{th} element of the story, choose a $x \times x$ square and move it by x squares, using the associated direction. Swap it with the square it replaces.
- Step 2 *S-box-website mapping*. Connect the story to the website to come up with a word/words $_R$. Convert letter values (mapping a=0, z=25) in the word(s) to integers, add a 0 to the number if it is a single-digit integer. Treat integers as (x,y) coordinates and find the corresponding characters in the table. Save this sequence of characters as the *password*. (For example: Connecting Tarzan to Amazon may result in the word “shirt” which maps to letter values “19 8 9 18 20”. Adding 0s to single digits, “19 80 90 18 20”, and mapping to the S-box results in coordinates (1,9), (8,0) etc. The password: v’tu.)

Song Password. This method relies on two sources of randomness – songs and a 4 digit key. s : A 4-digit pin. $h_R(s, w)$:

- Step 1 Reduce w to a 4 letter mnemonic. (*Flipkart* becomes *f p k t*)
- Step 2 Choose a 4 digit key $_R$. (3 8 1 9)
- Step 3 Choose 4 songs $_R$ starting from each letter of the mnemonic. These should be songs (not necessarily in English!) that have significance or are easy to remember. (*Fade*, *Panama*, *King of Mars* and *Teddy Boy*.)
- Step 4 Choose words $_R$ from each song, corresponding to each digit of the key, and concatenate to form a *Song String*, S_x . (3rd word from *Fade*, 8th word from *Panama*, 1st word from *King of Mars* and 9th word from *Teddy Boy*.)
- Step 5 After every vowel in S_x , insert a special character closest $_R$ to the vowel on the keyboard. If there is more than 1 special character equidistant from the vowel, choose $_R$ one and remember it. (For *o*, ‘(or ’), for *e* ‘\$’ or ‘#’.)
- Step 6 Choose three characters $_R$ (letters or symbols) and move them to the end of the password. Repeat with another group of three. Then remove every alternate character (starting with the first). *password*: resultant string.

Sources of randomness: Interpretations of linguistic fillers as words, choice of special character and characters to move.

Internal Sentence. s : A rarely used word $_R$ from any language. $h_R(s, w)$: Create a sentence connecting the website to the word. *password*: Sentence created.

5 Analysis of Hash Functions

This section analyzes the security and real-world effectiveness of our hash functions via several metrics, including a user study: 134 individuals aged 18–25 were surveyed, with each user generating passwords using 2 different randomly-assigned algorithms. Each algorithm had an average of 56 responses. We also include Cue-Pin-Select [4] in our survey.

5.1 Generation and Retention

Previous attempts have suggested “intolerably slow” methods [11]. Our protocols can be executed by the average user within 5 min for generation, and recollection time decreases significantly with repetition. The key human-computability properties of F_R are: (1) Reliance on cognitive and visual cues for stable, rapid recall⁸ (2) Minimal effort, and limited access to education or writing resources.

Some of our methods retain significant security without access to any external materials for generation. The Memory Palace and Internal protocols need only a keyboard (or pictures of standard keyboards; no writing materials or internet, though access to these would decrease cognitive load).

The ability to recall or regenerate a password is essential to its effective security; lower memorability leads to frequent passwords resets and frustration that may lead to users abandoning the algorithm. Users were surveyed over a week to test password retention. See Fig. 3 and Table 2. Methods with less successful recall (Cue-Pin-Select, Song password and Scrambled box) seem to require more explicit memorization. Associative techniques can exponentially increase ease of password recollection (Memory Palace, Internal Sentence), and provably improve system security [9]. Therefore we recommend the use of partial visual cues for subkey association whenever possible.

Table 2. R: Recall/regeneration of passwords. Attempts: Number of people who attempted R. Total R: exact recall/regeneration of 1 or more passwords created.

<i>Hashing algorithm</i>	<i>Attempts</i>	Password Memorability	
		<i>Complete R</i>	<i>Partial R</i>
Internal Sentence	42	21 (50%)	7 (17%)
Memory Palace	45	19 (43%)	6 (14%)
Song words	42	10 (24%)	11 (27%)
Cue Pin Select	47	11 (24%)	5 (11%)
Scrambled box	29	6 (21%)	4 (14%)

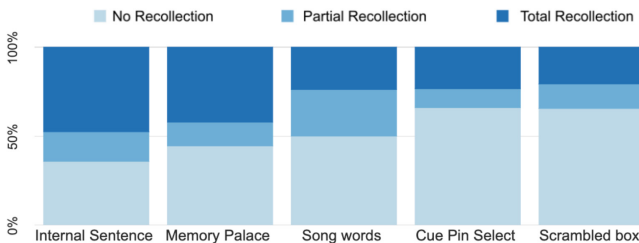


Fig. 3. Password recollection visualized (based on Table 2)

⁸ Some of which are proven to last in memory 17 years without repeated rehearsal [11].

The rightmost area of Fig. 4 indicates perfectly recalled passwords, with larger bubbles indicating a more significant percentage of users with perfect recollection. Ideal functions are large bubbles at the rightmost end of the graph with an average password length above 10 characters (see Sect. 3).

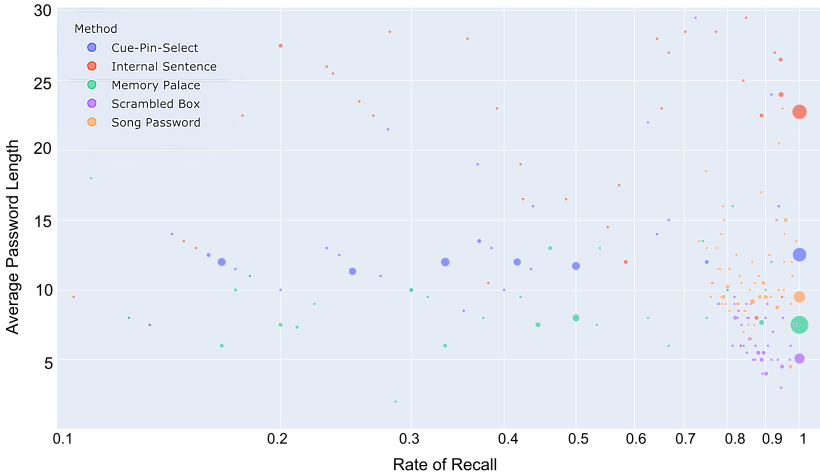


Fig. 4. Bubble chart of the rate of password recollection for hash functions. Each function is represented by a color; the frequency of each rate of recall (recall measured by: $\mathbb{S}(p_i, p_r)$ where \mathbb{S} corresponds to the Gestalt Pattern Matching (Ratcliff/Obershelp string similarity algorithm [26, 28]) corresponds to the size of each bubble, p_i is the initial password and p_r is the remembered password; the axes measure password length and the frequency of each length. (Color figure online)

Each time a password is recalled using a key, a user-familiar memory (object, space, color etc.) is associated with the key. This key-memory association is repeated until thinking of one automatically brings the other to mind [23]. We emphasize that, as in all reasonable systems, the generation method is public, and the only secret that needs to be remembered is this key.

The advantage of involving the methods proposed in this paper (such as visual, associative, implicit memory) is that they can be adapted to existing password generation methods. *E.g.*, Cue-Pin-Select can be modified to choose random words with visual or associative cues drawing on implicit memory.

5.2 Effective Security

We propose the concept of *effective security*. A password generation scheme may be incredibly secure, but is useless⁹ if it is so hard that most users just

⁹ Assuming an appropriate threat actor – imagining an adversarial ‘evil’ sibling with occasional read-only access to your living space is a useful rule of thumb.

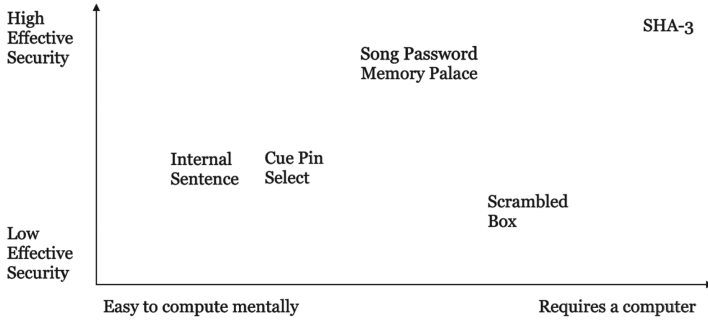


Fig. 5. Mapping effective security (password security and user comfort with algorithms) and ease of use (user perception on a scale of requiring no resources, to requiring computers). *Axes are exaggerated subjectively for illustration.*

write down their passwords. (See Fig. 5.) The effective security of a function F_R is the actual difficulty of breaking one of its assumptions in real-world use by laypersons. The ideal human-computable hash function is easy enough (and grows easier through repeated use) to encourage humans to use it, while retaining the necessary entropy to ensure security by resisting attacks.

Traditional cryptographic evaluations are built to evaluate functions designed for computers. We present a range of strategies for security evaluation in Appendix A. These strategies are **not** indicative of security by themselves, but taken in combination provide a good measure of the relative security of each function; further work is required to understand the security of such methods.

5.3 User Study and Improvements

We perform a survey comprising $n = 134$ individuals, with an average of 56 users suggesting improvements for each algorithm. We present baseline entropy evaluations for each function¹⁰, measure passwords from each function against current security standards and suggest improvements based on user feedback.

Our human computable hash functions average a password entropy of 78.07 bits, significantly higher than the average entropy of 40.54 bits per password as estimated by Microsoft [13]. These functions also encourage higher entropy by increasing use and distribution of symbols and capitalization. Memory Palace, Song Password, and Scrambled Box increase the number of symbols per average password to 3.188 symbols, compared to a baseline of 0.2 symbols. Capital letters decrease to 0.412, lower than 1.1 without hash functions. However, as evident in Fig. 7, capitalization is more distributed across location, rather than concentrated towards the first character of the password [20].

¹⁰ Assuming character entropies are independent. We do not consider dictionary attacks, character frequencies etc. as these would require a large number of passwords to be statistically valid, and due to unique user memory configurations R we cannot computationally generate large numbers of passwords.

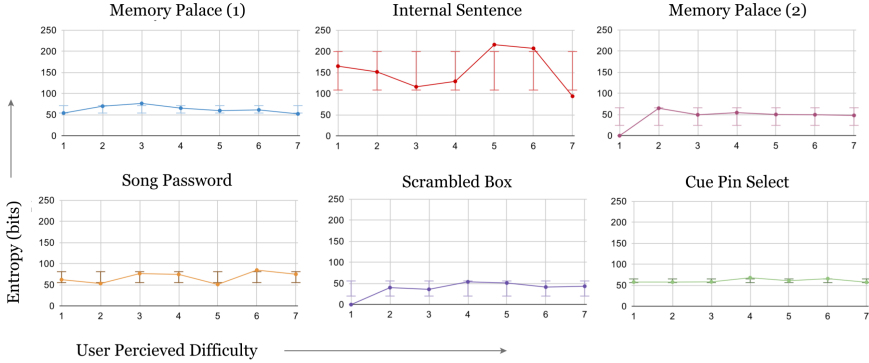


Fig. 6. Each point represents the mean entropy of passwords with some user perceived difficulty (std. dev. error bars). Memory Palace Step 1 was presented as “method 1” to users; 2 included all steps described in Sect. 4.1. X-axis: User Perceived Difficulty; Y-axis: Password Entropy.

Our results are reasonably representative of the general population of password users [24]. Our choice of sample size is based on [21] and [24]. Our sample is drawn from students in a medium-sized university in India and may be applicable to similar demographic profiles. In addition, the sample represents a range of language, educational and income backgrounds. However, the proportions of these demographics are not the same as the general population. Beyond the obvious age bias (college-aged individuals), the sample is biased towards individuals willing to participate in the survey in exchange for food and money (both standardized), and all data is self-reported. In addition, Cochran’s formula [22] recommends a sample size of 100 individuals based on the proportion of internet users in the world (53.6% of the global population in 2019 [7]), a 95% confidence interval and a 10% error margin. Compared to previous human computable password research [4], we use a significantly larger sample size with a more representative demographic of password users. Thus, our results, extrapolated prudently, can apply to the broader population.

Scrambled Box and Song require writing (the latter requires access to a music repository) and are harder than the first two methods for users. Song and

Table 3. Graceful degradation, mean entropies, and their standard deviations.

<i>Function</i>	<i>Mean entropy (bits)</i>	Graceful degradation and Entropy	
		<i>Standard Deviation</i>	<i>Graceful Degradation</i>
Internal Method	153.95	97.14	0.66
Memory Palace	51.08	25.84	0.38
Song Words	74.57	44.12	0.49
Cue Pin Select	61.96	17.62	1.06
Scrambled Box	45.15	33.15	0.84

Cue-Pin-Select also require greater intermediate key generation – choosing and explicitly recalling random unique words/songs and a pin/word. Comparatively, Internal Sentence and Memory Palace use associations already familiar to users.

Graceful degradation in Table 3 measures increase in difficulty with decrease in education levels. Larger graceful degradation corresponds to functions that require higher education levels.

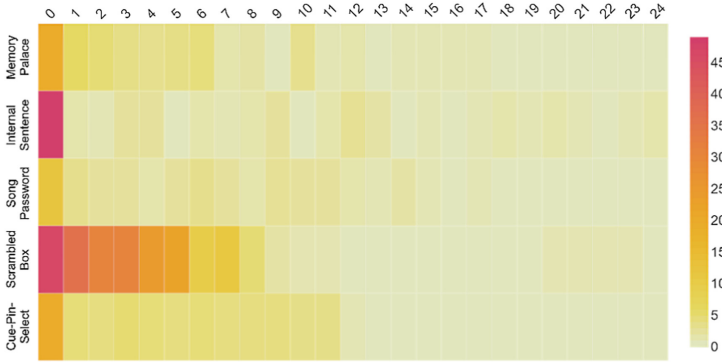


Fig. 7. Heatmap of the incidence of capital letters at different indexes. Passwords >25 characters are omitted

Memory Palace. With the aid of partial visual cues, memorizing hundreds of cues for subkey generation (objects, areas, memories etc.) [17] is unnecessary and the user can focus on subkey-cue associations. Most keys were in 4% of the 100 most common words in English, including references to common household objects and local languages. After hashing subkeys with each website, no English words were identifiable (excluding users who misinterpreted instructions).

Users were satisfied with the security but suggested clearer navigational guidance. A common struggle was navigating dead-ends with visually unremarkable cues. A significant proportion of users struggled with Step 2 and favored Step 1 and 3. Some users stated they would adapt Step 1 for future password generation.

Scrambled Box. The key is the ‘box’ of pseudo-randomly scrambled symbols. This can be written down or shared, but must be unique to each user, who only needs to remember website-subkey associations. Users found rearranging symbols hard and preferred fewer instructions, but liked the lack of memorization (Table 4).

Song Password. This scheme amplifies randomness in the input. For example, using songs: *Fade*, *Panama*, *King of Mars*, *Teddy Boy* with user one’s $PIN_1 = 3819$ and user 2’s $PIN_2 = 7144$, passwords generated are `mse$(i(o)*` and `tsto)mhS` (a similarity of 0.33% [26, 28]).

Users struggled with pins and associating different songs. Some users preferred not to remove or shift alternate characters, while others remarked they would adapt this method for future password generation.

Table 4. Security refers to the %age of passwords with ≥ 1 Number or Symbol. Length and difficulty are averages. Difficulty was assessed by users.

	<i>password length</i>	Survey Results	
		<i>security</i>	<i>difficulty (1-7)</i>
Internal Sentence	25.91	9.90	2.52
Memory Palace	8.42	86.06	5.38
Song Words	11.50	92.16	5.41
Cue Pin Select	12.29	3.21	4.44
Scrambled Box	6.71	94.11	5.68

Internal Sentence. Users preferred this method for ease of use but struggled with remembering word order, verb and adjective choice, etc. or found passwords generated too long to recall. Users felt this method was insecure as it did not generate special characters or capitalization. The entropy for this method is misleading, as passwords often contain words susceptible to dictionary attacks.

Cue-Pin-Select. Word and pin recollection were challenging, users preferred associating words with cues over random cues, and suggested reducing the number of random words from 6 to 4. In general users requested stronger associative and implicit memory modifications to the method. Across all passwords and algorithms mentioned in 4.1, average password entropy is 78.07 bits and average password length is 11.83 characters (i.e. numbers, symbols and letters) (Fig. 8).

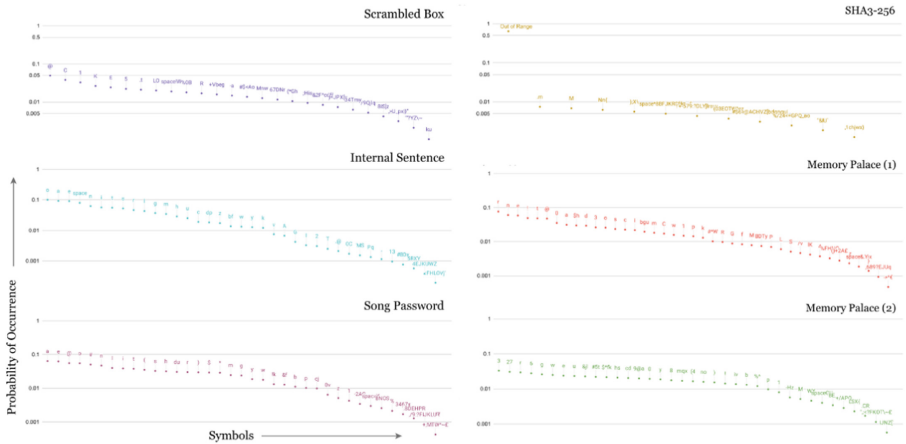


Fig. 8. Symbol occurrence by method. Y-axis: log scale. X-axis based on symbol rank (most to least probable). SHA3-256 hashes were converted to latin-1 encoding to get typable character frequencies [29]. Memory Palace as in Fig. 6.

5.4 Machine-Learning Based Analysis Using LSTMs

A simple machine learning system was used to predict the k^{th} character given the previous $k - 1$ characters of the password to further evaluate randomness. This is based on a long line of research starting from Shannon’s entropy experiment [18]. We used all characters except the last for training (see Table 5).

Table 5. We used a Long Short Term Memory Network [33] to learn dependencies. The 50-cell LSTM was tested with two trials of 100 and 200 epochs.

<i>Scheme</i>	<i>100 epochs</i>	Testing Accuracy (in %)
		<i>200 epochs</i>
Internal Method	53.13	58.41
Memory Palace	18.42	19.91
Song Words	21.71	23.37
Cue Pin Select	47.56	46.76
Scrambled Box	29.31	28.44

6 Real-World Password Generation Methods

How do people currently generate (and remember) passwords? Our survey suggests that people use a combination of words, followed by digits and symbols (in that order), indicating construction in order of ease of recollection. Common associations: names of relatives, fictional characters, nicknames, etc.; digits or symbols—birth dates, reversed phone numbers, even credit card numbers!

Some used inventive techniques to balance security with memorability: account expiry dates, rhymes, snacks and manufacturing dates, and slang words. Several users reused passwords with the awareness of compromised security, citing a lack of convenient options. A small population added random words from different languages. (Full database of results omitted for brevity.)

We observed that users designed passwords with human adversaries in mind and thus mistakenly believed that using animals or objects they disliked, using common character substitutions for letters (“leetspeak”), or misspelling words created a secure password. Based on previous work [44] and our survey, we recommend all platforms with password requirements brief users on current strategies used by computationally-equipped adversaries, such as dictionary attacks, frequency analysis etc. to reduce the usage of insecure passwords.

7 Conclusion

We propose a range of human-computable hashing algorithms with string and non-string inputs, designed for password generation and management. We exploit

users’ unique memory configurations to drive our design, drawing upon existing neuroscientific research. We also collate current password advice across hundreds of popular websites and applications, and survey users on their current password generation methods, highlighting major issues and discussing mitigation.

Our functions are validated and tested using a survey ($n = 134$) to understand real-world usability. We note that larger surveys across a range of age groups are required to better classify the security and usability implications. Further work also needs to be done to explore the kinds of atomic human-computed operations that produce stable output useful for cryptography.

A Cryptographic Security

Given the limitations imposed by the very nature of algorithms optimized for humans (which are intentionally difficult to represent on a computer) these methods cannot be used directly; we use approximate, illustrative calculations to indicate the likelihood of a given scheme satisfying some property.

When an adversary attempts to guess a user’s password for random accounts after seeing m/λ other random (account, password) pairs for the same user, a hash function h_R is considered UF-RCA (Unforgeability Under Random Challenge Attack) secure if a poly-time adversary can guess a new (account, password) pair with negligible success probability. [6]

For any hash function $h_R(s, w_i) \rightarrow y_i$ the adversary attempts to either guess s , or guess y_j for some w_j , based on knowledge of $C = \{(y_1, w_1), (y_2, w_2), \dots, (y_n, w_n)\}$ where $(y_j, w_j) \notin C$. The probability of correctly guessing the output (hash) for website w_j without knowing s , i.e., $P((y_j, w_j) | y_j = h_R(s, w_j) \wedge (y_j, w_j) \notin C) \leq \epsilon$ for any probabilistic polynomial time adversary.

A.1 Pre-image Resistance

Given only h_R (public hash function) and $h_R(w, s_k)$ (a password), pre-image resistance requires that it must be computationally hard to deduce s_k , the subkey, and s , the master secret. Note that R is unclonable in our setup.

Memory Palace: Given the hash, every alternate letter is either (Sect. 4.1):

- $l = \mathbb{S}(x, y)$ where \mathbb{S} : sum and x, y are two letters
- a diagonal mapping of l on the keyboard

Every letter l in the subkey depends on two other letters x, y such that¹¹:

$$L \begin{cases} x + y & \text{for } x + y < 26 \\ x + y - 26 & \text{for } x + y \geq 26 \end{cases}$$

¹¹ Assuming the alphabet is indexed from 0.

The probability of guessing x and y given l is $P(x, y|l) \leq \frac{1}{13}(0.0769)$ or $\frac{1}{14}(0.0714)$ based on 13–14 pairs of $s(x, y)$ for every l . This reveals nothing about the permutation, e.g., $a_i + b_i = b_i + a_i = c_i$, where a_i is the index of the letter a . In this case both ab and ba are candidate permutations for c , as are 13 other letter-pairs such as cz , *no* etc. So, every character of the hash depends on several possible letter-pairs in the previous text (confusion). Taking into account letter-pair permutations, the probability space increases such that: $P(x, y|l) \leq \frac{1}{26}(0.0385)$ or $\frac{1}{28}(0.0357)$. The adversary now guesses the underlying letters with $\leq 4\%$ probability. If all (x, y) and their permutations are discovered, the user’s subkey is discovered. However this does not reveal other subkeys due to sources of randomness within the function, as elaborated in Appendix A.

Song Password: Passwords generated by this method had no identifiable words from the English language, or local languages. The title word of the song for the examples used in Step 3 in the description of Song Password formed a maximum of 10% of the song lyrics. An adversary has to undo several layers of confusion based on R , such as shifting characters to different positions, removing characters etc., which leave no identifiable words from the English language, or local languages in the final password, to deduce s from the hash. It is also computationally hard to predict characters that may have been removed due to character shifts before deletion that do not preserve letter frequencies or word patterns.

Scrambled Box This method is strongly resistant to pre-image attacks (a public S-box degrades gracefully). Given the S-box and the password, each character c in the S-box corresponds to a unique coordinate set (x, y) which in turn is the index xy of an alphabet. If the letter maps to a single-digit index, y may be a digit from the index of the next alphabet. Due to the vast number of possibilities for each character mapping in the password, we propose that finding s given the user’s S-box, w and $h(s, w)$, is computationally infeasible.

Internal Sentence: Here, $s_k = s$ is a “unique” word, and $h_R(s, w)$ is a sentence including s and w . A frequency analysis of words will suggest a candidate s , and w is publicly known. Passwords resulting from this hashing method carried high entropy, but most passwords (138/202) with 4–17 words, included between 1–12 words from the 3000 most frequently used English words [27] and thus are not UF-RCA secure, as with n (account, password) pairs for the same user, a “unique” s can be deduced with word frequency analysis. Combined with word permutations a large number of candidate passwords can be produced with negligible computational effort. However this method is still weakly collision-free - long sentences without specified one-way mappings of (subkey \rightarrow word combinations) result in a low incidence of $h_R(m') = h_R(m)$.

A.2 Collision Resistance and Randomness

An adversary cannot even compute or verify h_R efficiently, since R is unique to each user. In that sense, our hash function is similar to a physically unclonable function [37]. Our analysis suggests that given a password y , guessing m , $P(h_R(m) = y) \leq \epsilon = 2^{-78}$ in the average case (length 11.83), as analysed at the

end of Sect. 5.3. (Most of our functions are also strongly collision free; details omitted for brevity.) We refer readers to [4] for the security of Cue-Pin-Select.

We observe a variety of sources of randomness for each R . Understanding and manipulating this randomness is an interesting problem for future research.

References

1. Alexa: The top 500 sites on the Web. <https://www.alexa.com/topsites>
2. Baddeley, A.D.: Human Memory: Theory and Practice. Psychology Press, London (1997)
3. BestReviews: Which password managers have been hacked? - Best reviews, July 2018. <https://password-managers.bestreviews.net/faq/which-password-managers-have-been-hacked/>
4. Blanchard, N., Gabasova, L., Selker, T., Sennesh., E.: Cue-Pin-Select, a Secure and Usable Offline Password Scheme (2018). fhal-01781231
5. Blocki, J., Blum, M., Datta, A.: Naturally rehearsing passwords. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8270, pp. 361–380. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42045-0_19
6. Blocki, J., Blum, M., Datta, A., Vempala, S.: Towards human computable passwords. arXiv preprint [arXiv:1404.0024](https://arxiv.org/abs/1404.0024) (2014)
7. Bogdan-Martin, D.: (2019). <https://www.itu.int/en/ITU-D/Statistics/Documents/facts/FactsFigures2019.pdf>
8. Buys, B.: Estimating password crack times. <https://www.betterbuys.com/estimating-password-cracking-times/>
9. Chakravarthy, A., et al.: A novel approach for password authentication using bidirectional associative memory. arXiv preprint [arXiv:1112.2265](https://arxiv.org/abs/1112.2265) (2011)
10. Cybersecurity Ventures: New report finds 300 billion passwords will be at risk by 2020 (2017). <https://cybersecurityventures.com/300-billion-passwords/>
11. Denning, T., Bowers, K., Van Dijk, M., Juels, A.: Exploring implicit memory for painless password recovery. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2615–2618 (2011)
12. Eastlake, C.: “Schiller.” Randomness requirements for security, June 2005. <https://tools.ietf.org/pdf/rfc4086.pdf>
13. Florencio, D., Herley, C.: A large-scale study of web password habits. In: Proceedings of the 16th International Conference on World Wide Web, pp. 657–666 (2007)
14. Fung, E.: Hash functions. <https://www.cs.usfca.edu/~ejung/courses/686/lectures/05hash.pdf>
15. Gedeon, K.: Popular password managers can get hacked: should you keep using them? March 2020. <https://www.laptopmag.com/news/popular-password-managers-can-get-hacked-should-you-keep-using-them>
16. Google, H.P.s.: Online security survey Google/Harris poll, February 2019. http://services.google.com/fh/files/blogs/google_security_infographic.pdf
17. Guardian, D.: Uncovering password habits: are users’ password security habits improving? (Infographic), December 2018. <https://digitalguardian.com/blog/uncovering-password-habits-are-users-password-security-habits-improving-infographic>
18. Hamid Moradi, J.W.G.B.: Entropy of English text (1998). <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.5610&rep=rep1&type=pdf>

19. Hardwick, J.: Top 100 most visited websites by search traffic (as of 2020), May 2020. <https://ahrefs.com/blog/most-visited-websites/>
20. Jonathan: Beyond password length and complexity, May 2019. <https://resources.infosecinstitute.com/beyond-password-length-complexity/#:~:text=PasswordLength,numbersand0.2specialcharacters>
21. Komanduri, S., et al.: Of passwords and people: measuring the effect of password-composition policies. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2595–2604 (2011)
22. Kotrlik, J., Higgins, C.: Organizational research: determining appropriate sample size in survey research appropriate sample size in survey research. *Inf. Technol. Learn. Perform. J.* **19**(1), 43 (2001)
23. Loterre. <https://www.loterre.fr/skosmos/P66/en/page/-SQ2MHWLN-Q>
24. Mazurek, M.L., et al.: Measuring password guessability for an entire university. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, pp. 173–186 (2013)
25. O’Flaherty, K.: Password managers have a security flaw - here’s how to avoid it, February 2019. <https://www.forbes.com/sites/kateoflahertyuk/2019/02/20/password-managers-have-a-security-flaw-heres-how-to-avoid-it/>
26. Paul, E.: Black. 2004. Ratcliff/obershelp pattern recognition. *Dictionary of Algorithms and Data Structures* 17 (2004)
27. Press, O.U.: The Oxford 3000. <https://www.oxfordlearnersdictionaries.com/about/oxford3000>
28. Python Software Foundation, P.S.F.: 7.4. difflib - helpers for computing deltas (2020). <https://docs.python.org/2/library/difflib.html>
29. Ruthu, R.: Github, code reference, July 2020. <https://github.com/debayanLab/trenchcoat>
30. Safe, S.: <https://splashdata.com/press/releases.htm>
31. Schacter, D.L., Chiu, C.Y.P., Ochsner, K.N.: Implicit memory: a selective review. *Ann. Rev. Neurosci.* **16**(1), 159–182 (1993)
32. Schneier, B.: (2014). https://www.schneier.com/blog/archives/2014/03/choosing-secure_1.html
33. Shi, Z., Shi, M., Li, C.: The prediction of character based on recurrent neural network language model. In: 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), pp. 613–616 (2017)
34. Smith, A.: Americans, password management and mobile security, August 2020. <https://www.pewresearch.org/internet/2017/01/26/2-password-management-and-mobile-security/>
35. Stats, D.: RSA Challenge, June 2020. http://stats.distributed.net/projects.php?project_id=8
36. Stolyar, B.: Apple unveils the most popular iphone apps of 2019, December 2019. <https://mashable.com/article/apple-most-popular-iphone-apps-2019/>
37. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: 2007 44th ACM/IEEE Design Automation Conference, pp. 9–14. IEEE (2007)
38. Toponce, A.: Strong passwords need entropy (2011). <https://pthree.org/2011/03/07/strong-passwords-need-entropy/>
39. Wikipedia contributors: list of most-downloaded Google play applications – Wikipedia, the free Encyclopedia (2020). https://en.wikipedia.org/w/index.php?title=List_of_most-downloaded_Google_Play_applications&oldid=962291709. Accessed 5 July 2020

40. Winder, D.: Ranked: the world's top 100 worst passwords (2019). <https://www.forbes.com/sites/daveywinder/2019/12/14/ranked-the-worlds-100-worst-passwords/#54064d4869b4>
41. WordFence: Password authentication and cracking, June 2018. <https://www.wordfence.com/learn/how-passwords-work-and-cracking-passwords/>
42. Yan, J., Blackwell, A., Anderson, R., Grant, A.: Password memorability and security: empirical results. *IEEE Secur. Priv.* **2**(5), 25–31 (2004)
43. Zetter, K.: It's insanely easy to hack hospital equipment, June 2017. <https://www.wired.com/2014/04/hospital-equipment-vulnerable/>
44. Zhang-Kennedy, L., Chiasson, S., Biddle, R.: Password advice shouldn't be boring: visualizing password guessing attacks. In: 2013 APWG eCrime Researchers Summit, pp. 1–11 (2013)



Provably Secure Scalable Distributed Authentication for Clouds

Andrea Huszti^(✉) and Norbert Oláh^(✉) 

Faculty of Informatics, University of Debrecen,
Kassai street 26., Debrecen 4028, Hungary
{huszti.andrea,olah.norbert}@inf.unideb.hu
<https://www.inf.unideb.hu/>

Abstract. One of the most used authentication methods is based on short secrets like password, where usually the hash of the secrets are stored in a central database. In case of server compromise the secrets are vulnerable to theft. A possible solution to this problem to apply distributed systems. We propose a mutual authentication protocol with key agreement, where identity verification is carried out by multiple servers applying secret sharing technology on server side. The protocol results in a session key which provides the confidentiality of the later messages between the participants. In our solution we also achieve robustness and scalability as well. To show that the proposed protocol is provably secure, we apply the threshold hybrid corruption model. We assume that among the randomly chosen k servers, there is always at least one uncorrupted and the authentication server reveals at most the long-lived keys. We prove that the protocol is secure in the random oracle model, if Message Authentication Code (MAC) is universally unforgeable under an adaptive chosen-message attack, the symmetric encryption scheme is indistinguishable under chosen plaintext attack, moreover Elliptic Curve Computational Diffie-Hellman assumption holds in the elliptic curve group.

Keywords: Authenticated key agreement · Provable security · Distributed system · Cloud authentication · Threshold hybrid corruption model

1 Introduction

1.1 Motivation and Related Work

A distributed system consists of multiple, autonomous computers that communicate through a network even during completing their task. The goal of a

This work was supported by the construction EFOP-3.6.2-16-2017-00015 and the SETIT Project (no. 2018-1.2.1-NKP-2018-00004), which has been implemented with support provided by the European Union, co-financed by the European Social Fund and the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

distributed system is to solve a single problem by breaking it down into several tasks where each task is computed by a computer of the system. Distributed systems can run in a cloud infrastructure as well. Secure user authentication is an important issue of cloud services. If it is breached, confidentiality and integrity of the data or services may be compromised. In the case of Software as a Service model, the cloud service provider takes responsibility for securing all the data from unauthorized access.

In practice, OpenStack [30] is one of the most popular cloud computing softwares. OpenStack Identity service supports multiple methods of authentication, including user name and password, Lightweight Directory Access Protocol (LDAP), and external authentication methods (i.e. Kerberos). There are fears about these systems due to their centralized structure. Services storing password information for a large number of enterprises in a central database are primary targets for hackers (e.g. Golden Ticket Attack [24,28], OneLogin attack [29]). The recent hack of OneLogin, an Identity and Access Management (IAM) provider proves that the credentials may not be as safe as we are led to believe. The weakest point of Kerberos authentication model is the key distribution center (KDC). The entire authentication system depends on the trustability of the KDC, so anyone who can compromise system security on a KDC system can theoretically compromise the authentication of all users of systems depending on the KDC. On the other hand, Kerberos requires the continuous availability of the KDC server, if the server is not available, no one can log in.

In scientific literatures, usually centralized, one-factor [17,23] or two-factor identity verification protocols [12,13] are proposed. Xavier Boyen presented a Hidden Credential Retrieval protocol [9], where the protocol applies one server solution and a blind signature technique with a user password. However, the concept of distributing authentication to multiple servers enhance the security level. The advantage of a distributed system is that external attackers have to attack multiple servers simultaneously, which increases the attack cost.

Several protocols are proposed for a multi-server environment. Sood, Sarje and Singh [33] and Brainard et al. [11] suggested authentication protocols in a two-server environment, where two servers together decided on the correctness of the password submitted for authentication. Katz et al. [21] demonstrated the first provably-secure two-server protocol for the important password-only setting (in which the user needs to remember only a password, and not the servers' public keys), and was the first two-server protocol (in any setting) with a proof of security in the standard model. Acar et al. constructed a solution [1] in which securely store the corresponding secret key and blinded by some function of user password at storage provider(s) different from the login server. In this proposition [16] a multiple-server authentication protocol is designed, where one-time passwords are shared among the cloud servers. A Merkle tree or a hash tree [26] is applied for verifying the correctness of the one-time password. Most of the cases after successful authentication an encrypted communication channel is established. Our goal is also to establish a symmetric encryption key.

Passwords are often used as authentication information in key exchange protocols. In case of a password-authenticated key exchange (PAKE) after user registration a user and a server establish a session key for a secure channel [6, 8, 10, 22, 25]. In a multiple-server environment usually threshold password-authenticated key exchange protocols are designed. Devriş Işler and Alptekin Küpçü introduced a scheme [19] where the protocol ensures that multiple storage providers can be employed, and the adversary must corrupt the login server and threshold-many storage providers to be able to mount an offline dictionary attack. Afterward, they proposed a new framework [20] for distributed single password protocols (DiSPP). In their protocol, the user stores the secret among storage providers (e.g. personal devices, online storage providers) and accesses it by using his/her password. There are n login servers and n storage providers, and to be able to mount an offline dictionary attack successfully, an adversary must corrupt t login servers in addition to t storage providers. Mario Di Raimondo and Rosario Gennaro recommended two threshold password authenticated key exchanges [31] where the protocols require $n > 3t$ servers to work. They enforce a transparency property: from the point of view of the client the protocol should look exactly like a centralized protocol of Katz, Ostrovsky, and Yung (KOY protocol). They proved that their protocol is provably secure if $n > 3t$ and the Decisional Diffie-Hellman Assumption holds. Password-Protected Secret Sharing (PPSS) scheme with parameters (t, n) was formalized by Bagherzandi et al. [2]. Jarecki et al. [18] present the first round-optimal PPSS scheme, requiring just one message from user to server and from server to user, and prove its security in the challenging password-only setting where users do not have access to an authenticated public key. The scheme uses an Oblivious Pseudorandom Function (OPRF) and builds the first single-round password-only Threshold-PAKE protocol in the Common Reference String (CRS) and random oracle models (ROM).

We propose a multi-server password-based authenticated key exchange scheme. It is similar to the (k, n) threshold PAKE systems, however we don't apply the secret-sharing algorithms. In contrast to other threshold password-based protocols applying secret-sharing algorithms [2, 6, 8, 10, 18, 20, 22, 25, 31], although we share the password information among the servers, it is not reconstructed from the shares to verify it. Ford et al. [14] proposed a protocol that securely generates a strong secret from a weak secret (password), based on communications exchanges with two or more independent servers. We demonstrate a new way of generating a strong secret (e.g. long-lived key) from a password, it is also suitable for scalability. Comparing to other schemes we also consider the scalability property that is one of the main requirements for clouds. In the Internet of Things (IoT) environment an authenticated key exchange (AKE) protocol is presented [32] on wireless sensor networks and they focus on the key shares and establish the authenticated key between Wireless sensor networks (WSNs) and the cloud server, which performs a centralization authentication. Another variant of AKE is demonstrated [34] which includes a permanent Control Server and cloud servers on 5G network. Our solution differs from these papers [32, 34]

since we can scale the generating long-lived keys on the user's and the provider's sides as well. Unlike [10, 14] we also provide a detailed security analysis based on the Bellare and Rogaway model.

1.2 Our Contribution

In this section, we give the details and the novelty of our solution, compared to the previous propositions. The main goal of *key exchange* protocols is to set up a shared secret key between two or more entities. In case of *key agreement*, both entities contribute to the joint secret key by providing information from which the key is derived. In mutual authentication parties who engage in a conversation in which each gains confidence that it is the other with whom he speaks. In protocols providing implicit key authentication, each participant is assured that no one other than the intended parties can learn the value of the session key. A key agreement protocol that provides mutual implicit key authentication is called an *authenticated key agreement* protocol (or AK protocol). A key agreement protocol provides key confirmation (of B to A) if A makes sure that B possesses the secret key. A protocol that provides mutual key authentication as well as mutual key confirmation is called an *authenticated key agreement with key confirmation* protocol (or an AKC protocol) [3].

Our main goal is to design an AKC protocol which takes advantage of distributed systems. Our protocol would fit into these systems and takes advantage of the capabilities of these systems like robustness, scalability and greater availability. We assume there are thousands of servers in a cloud system therefore we reject the single-server authentication (e.g. Kerberos) and instead we propose the multi-server authentication. It is important to note that a single point of failure occurs typically in single-server solutions. If the server is unavailable, the provider usually needs to ensure replication to tackle the failure of their servers. Our scheme consists of n servers and the user randomly selects $k \leq n$ ones for each authentication. Besides the randomly chosen k servers a server called authentication server is also chosen randomly in our solution. In the authentication phase, the k servers use their long-lived keys and the user's authentication is verified by the chosen server via the correct MAC values. Even if one or more servers fail out of the n servers, the client can still choose k servers randomly. So if one or more servers break down or become corrupt, the service provider will be able to service and authenticate the users securely.

During authentication, instead of securely constructing the secret password from its shares, a random challenge generated by the client is constructed and verified by the authentication server. The randomly chosen participating servers are able to compute their challenge shares with the help of the password-based long-lived key set during registration and send them to the authentication server. In this way, the confidentiality of the password is assured. We focused heavily on making the protocol effective and we achieved promising efficiency results. The related protocols in the literature in several cases employ asymmetric cryptographic primitives which are considered slow compared to symmetric solutions and hash functions. The results of our protocol can be led back to the facts

that the session key is generated by ECDH key exchange, moreover MAC, xor operations and symmetric encryption are applied.

Finally, we also provide the security proof of the protocol and we demonstrate that the protocol is provably secure. For our protocol, we extended the Bellare and Rogaway security model in [7] to prove that our multi-device scheme is secure and we introduced the threshold hybrid corruption model. We assume that among the randomly chosen k servers, there is always at least one uncorrupted and the authentication server reveals at most the long-lived keys. We prove that the proposed protocol is a secure AKC protocol in the random oracle model, assuming the ECCDH assumption holds in the elliptic curve group, if MAC is universally unforgeable under an adaptive chosen-message attack and the symmetric encryption scheme is indistinguishable under chosen plaintext attack.

1.3 Outline of Article

In Sect. 2, we describe the necessary preliminaries. In Sect. 3, we give the steps of the protocol. A security analysis is detailed, which includes the security model with the security requirements are given in Sect. 4. We formalize the adversarial model, and give the security proof and practical issues. In Sect. 5, our conclusion is given.

2 Preliminaries

Before we detail the protocol and the related security properties, the necessary security assumptions for the basic primitives are given.

Definition 1. *A message authentication code (or MAC) is a tuple of polynomial-time algorithms (Key_M, Mac, Ver) such that:*

1. *The key-generation algorithm Key_M takes as input the security parameter 1^κ and outputs a key K with $|K| \geq \kappa$. Key_M is probabilistic.*
2. *The tag-generation algorithm Mac takes as input a key K and a message $m \in \{0, 1\}^*$, and outputs a tag t . We write this as $t := Mac_K(m)$. We assume that Mac is deterministic.*
3. *The verification algorithm Ver takes as input a key K , a message m , and a tag t . It outputs a bit b , with $b = 1$ meaning valid and $b = 0$ meaning invalid. We assume without loss of generality that Ver is deterministic, and so write this as $b := Ver(m, t)$.*

Consider the experiment for a message authentication code (Key_M, Mac, Ver) , an adversary \mathcal{A} , and security parameter κ , as follows. The message authentication experiment $Exp_{Mac}^{forge}(\mathcal{A})$:

1. A random key K is generated by running $Key_M(1^\kappa)$.
2. The adversary \mathcal{A} is given input 1^κ and oracle access to $Mac_K(\cdot)$. The adversary eventually outputs a pair (m, t) .

3. The output of the experiment is defined to be 1 if and only if $Ver_K(m, t) = 1$ and m was never asked from the oracle $Mac_K(\cdot)$ before.

Definition 2. A message authentication code (Key_M, Mac, Ver) is existentially unforgeable under an adaptive chosen-message attack, if for all probabilistic polynomial-time adversaries \mathcal{A} , $Pr[Exp_{Mac}^{eforge}(\mathcal{A}) = 1]$ is negligible.

We apply elliptic curve cryptography to make our protocol suitable for resource constrained environment as well.

Definition 3. Consider an elliptic curve E defined over a finite field \mathbb{F}_q , a point $G \in E(\mathbb{F}_q)$ of order n . The Elliptic Curve Computational Diffie-Hellman (ECCDH) function $ECCDH(\cdot, \cdot)$ takes as input a pair of elements $(X, Y) \in \langle G \rangle^2$ and returns $dlog(X)dlog(Y)G$. We say that $\langle G \rangle$ satisfies the ECCDH assumption if for all probabilistic polynomial-time algorithm S , the probability that for a random element $(X, Y) \in \langle G \rangle^2$, S correctly returns $ECCDH(X, Y)$ is negligible.

Servers should be able to use a secret channel to exchange data in an encrypted manner.

Definition 4. A symmetric encryption scheme is a tuple of probabilistic polynomial-time algorithms (Key_E, Enc, Dec) .

1. The key-generation algorithm Key_E takes as input the security parameter 1^κ and outputs a random key $K \in \{0, 1\}^\kappa$.
2. Enc is an encryption algorithm that takes inputs key K and plaintext $m \in \{0, 1\}^*$, and outputs a ciphertext c .
3. Dec is a deterministic decryption algorithm that takes inputs key K and ciphertext c , and outputs the plaintext m .

To define secure encryption, let \mathcal{A} be an adversary and consider the experiment $Exp_{Enc}^{ind-cpa}(\mathcal{A})$ as follows.

1. A random key K is generated by running $Key_E(1^\kappa)$.
2. The adversary \mathcal{A} is given input 1^κ and oracle access to $Enc_K(\cdot)$, and \mathcal{A} outputs $m_0, m_1 \in \{0, 1\}^*$ with $|m_0| = |m_1|$.
3. A random bit b is chosen and a ciphertext $Enc_K(m_b)$ is computed and given to \mathcal{A} .
4. \mathcal{A} continues to have oracle access to $Enc_K(\cdot)$, and outputs a bit b' .
5. We define \mathcal{A} 's advantage to be $Adv_{Enc}^{ind-cpa}(\mathcal{A}) = |Pr[b' = b] - 1/2|$.

Definition 5. We say that a symmetric encryption scheme is indistinguishable under chosen plaintext attack if $Adv_{Enc}^{ind-cpa}(\mathcal{A})$ is negligible in κ for any efficient adversary \mathcal{A} .

3 The Proposed Scheme

In this section we propose a multi-server authenticated key agreement protocol with key confirmation. We assume that a secret symmetric, long-lived key is exchanged between each client and server during client registration. In the authentication phase, a client chooses several servers randomly to participate. The client verifies the identity of each server and one of the randomly chosen servers, the authentication server, proceeds the steps of the client authentication. Mutual authentication of the participants is based on the correctness of a calculation, where the secret, long-lived symmetric key is used. Besides the mutual authentication of the participants a secret, session key is exchanged between the client and the authentication server.

During registration, the client sets password-based long-lived keys with all the n servers. In such a system, in addition to the aspect of robustness, the property of scalability is also important. To achieve this, we propose a simple solution in which the client accesses the long-lived keys by using a password. We assume that a client software is running on the client device (e.g. smartcard, mobile phone etc.) that requires a password from the user to initiate the authentication process. After the client gives the password the client software generates the long-lived keys and the execution of authentication begins. The correctness of the password is verified by the servers not the client software, hence a client device does not store any information about the password.

During authentication a server only with the knowledge of the symmetric, long-lived key K_i , where $i \in \{1, \dots, k\}$ generated from the client password, is able to calculate the challenge value given by the client. The authentication server authenticates the client by verifying the correctness of all the k challenge values received from the participating servers.

In the proposed protocol servers communicate on secure channels to each other. We prefer one server chosen randomly that communicates to the client, hence the client does not need to communicate to all the k servers in parallel and build secure channels.

During the design of the protocol, the efficiency of authentication is ensured by MAC and other fast cryptographic algorithms (hash, xor operation, symmetric encryption). The protocol is provably secure and the necessary model and the formal proof are given. We apply distributed authentication, thus we extend the model with the concept of threshold hybrid corruption.

3.1 Setup

In the setup phase the system parameters are generated and the registration is executed between the client and the servers. We differentiate two participants: A *client* (I) asks for services and the *servers* (J_1, \dots, J_n). We denote by $\{0, 1\}^*$ the set of all binary strings of finite length. If x, y are strings, then $x||y$ denotes the concatenation of x and y . Let \oplus denote an exclusive or calculation. During setup all system parameters and keys are generated. Let E denote an elliptic curve defined over a finite field \mathbb{F}_q and $G \in E(\mathbb{F}_q)$ a point of

order \mathbf{n} . Elliptic curve parameters are chosen in a way that the system resists all known attacks on the elliptic curve discrete logarithm problem in $\langle G \rangle$. Let σ denote the length of an elliptic curve point binary representation. We also represent J_i , $i \in \{1, \dots, n\}$ as a bitstring with length σ . System parameters par are given by $par = (E, q, \mathbf{n}, G, H, H_0, Mac)$, where $H : \{0, 1\}^* \rightarrow \{0, 1\}^\nu$, $H_0 : \{0, 1\}^* \rightarrow \{0, 1\}^\iota$ are cryptographic hash functions and ν, ι are not necessarily different, ι is the size of the secret session key being exchanged. $Mac : \{0, 1\}^* \rightarrow \{0, 1\}^\nu$ is a MAC function. System parameters are publicly known. Long-lived secret, password-based symmetric keys (K_1, \dots, K_n) between the client and each server are exchanged securely. To provide message confidentiality between the servers each server possesses $n - 1$ symmetric encryption keys $(\overline{K}_1, \dots, \overline{K}_{n-1})$ for secure communication. These keys are short-term and exchanged securely.

3.2 Scalability

In this section, we present an algorithm providing scalability of our protocol. Let $KKDF$ denote a Keyed Key Derivation Function that for a message m and a key generates a secret key K , i.e. $K = KKDF_{key}(m)$. Let the message be the password psw and the $key = H(salt||psw)$, and c be the number of iteration. Value $KKDF_{key}(psw)$, and secret shares

$$KKDF_{key}^c(psw), KKDF_{key}^{c+1}(psw), \dots, KKDF_{key}^{c+n-2}(psw)$$

are calculated for the $n - 1$ servers. Finally let

$$K_n = KKDF_{key}(psw) \oplus KKDF_{key}^c(psw) \oplus \dots \oplus KKDF_{key}^{c+n-2}(psw)$$

for the n th server. If the number of servers are increased in the cloud, we take the secret part of one of the servers and divide it into as many parts as the number of new servers we want to add to the system. In general, for increasing n servers with k new ones, a $KKDF_{key}^l(psw)$ is chosen to be scaled and $KKDF_{key}^t(psw)$, where $t = c + n, \dots, c + n + k - 1$ are calculated. The secret share for the chosen server is modified to

$$K_{new} = KKDF_{key}^l(psw) \oplus KKDF_{key}^{c+n}(psw) \oplus \dots \oplus KKDF_{key}^{c+n+k-1}(psw).$$

The device stores for each server a list of numbers of iteration. The list has either one element, or if it is scaled by k new ones, than $k + 1$ elements. If we decrease the number of servers, we take the iteration numbers of the deleted servers and add them to the stored list of a remaining server. Observe, that to calculate a secret share from another share the key of the KKDF, the salt and the password are needed. Hence even if we scale a corrupted share, the new shares cannot be calculated. The salt is stored only on the client device and the password is known only by the client.

3.3 Authentication Phase

In the authentication phase, we utilize the benefits of the distributed system to perform multiple server authentication. In this phase, mutual authentication between a client and the randomly chosen servers is processed with a key agreement. Client I randomly chooses k servers out of the n . Server J_i , where $i = 1, \dots, k$ verifies whether I possesses the long-lived symmetric key K_i . At the end of the authentication, a secret session key ssk is exchanged. Figure 1 shows the process of authentication.

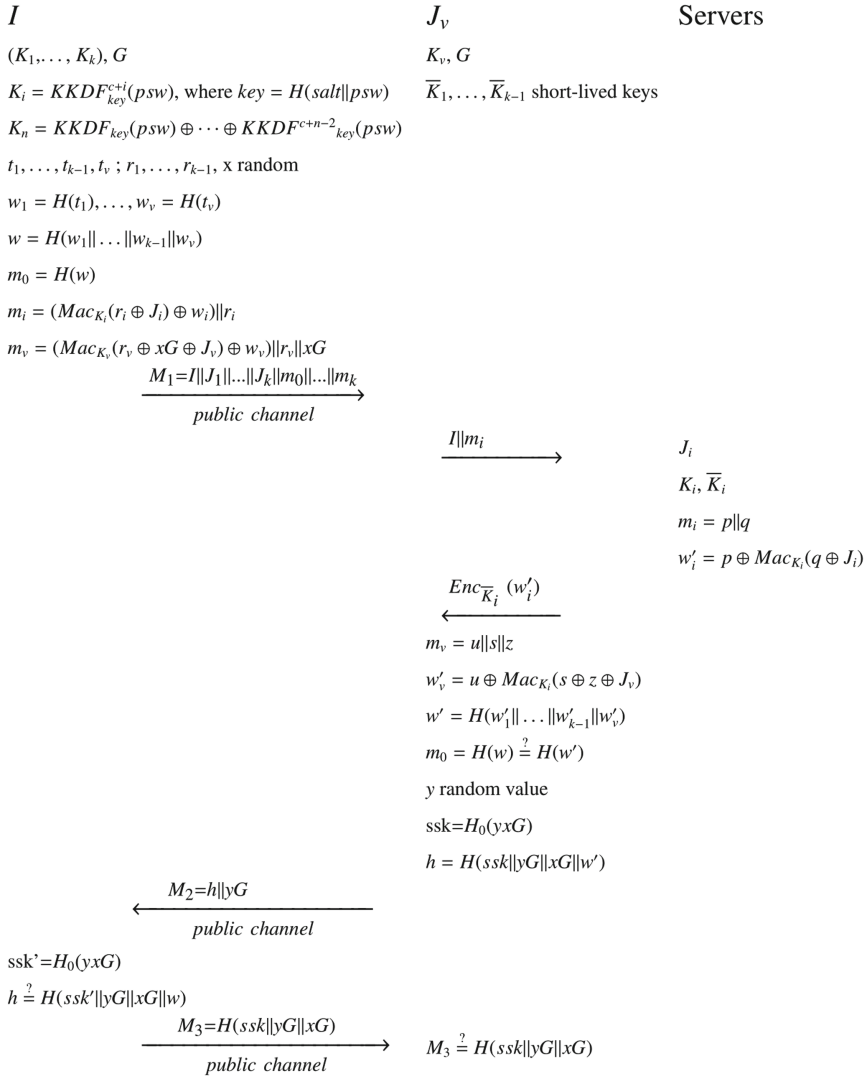


Fig. 1. Authentication

As the first step of the authentication, I selects k servers that are involved in the authentication. A random value $v \in \{1, \dots, k\}$ is generated by a time-based pseudorandom number generator. Let J_v denote the authentication server, which performs the authentication on server side. J_v communicates with the client and the other $k - 1$ servers. After entering the correct password, the client software calculates the long-lived keys from the password and the salt that is stored on the device. I generates random bitstrings t_1, \dots, t_k and calculates hash values $w_1 = H(t_1), \dots, w_k = H(t_k)$ and w , such that $w = H(w_1 || w_2 || \dots || w_k)$. Random values $r_i \in \{0, 1\}^\sigma$, where $i = 1, \dots, k$ are generated, too. Subsequently, I creates the first message M_1 , which is sent to the chosen authentication server J_v . I computes $m_0 = H(w)$ and $m_v = (Mac_{K_v}(r_v \oplus xG \oplus J_v) \oplus w_v) || r_v || xG$, where MAC is calculated with the long-lived symmetric key K_v . Moreover, m_v comprises xG , which is an elliptic curve point represented by a bitstring that is necessary for the key agreement, it is the client message of the elliptic curve Diffie-Hellman key exchange. The first message also consists of $m_i = (Mac_{K_i}(r_i \oplus J_i) \oplus w_i) || r_i$. Authentication of the participants is based on the correct calculation of the MAC values.

J_v receives message M_1 and forwards each m_i together with I to server J_i . Each server receives $I || m_i$, where $m_i = p || q$. Server J_i calculates $Mac_{K_i}(q \oplus J_i)$, where K_i is the long-lived key exchanged between the client and the server. Each server calculates $w'_i = p \oplus Mac_{K_i}(q \oplus J_i)$. Therefore, a server is able to calculate a valid w'_i only with the knowledge of K_i , the secret, long-lived key exchanged with I before. Value w'_i is sent back to J_v encrypted. J_v calculates w' from all w'_i received, and checks whether $H(w) = H(w')$ holds. If they are equal, then J_v makes sure about the authenticity of the client.

Thereafter J_v generates a random value $y \in \mathbb{Z}_n^*$ and computes the secret session key $ssk = H_0(yxG)$. J_v calculates response $M_2 = h || yG$, where $h = H(ssk || yG || xG || w)$ and yG is the server message of the EC Diffie-Hellman key exchange.

I receives $M_2 = h || yG$ and calculates the secret session key $ssk = H_0(yxG)$ and $h' = H(ssk || yG || xG || w)$. If $h = h'$, then I is confirmed that server J_v knows the secret session key, and the randomly chosen servers know the secret long-lived keys, hence their identity is verified.

As a last step $M_3 = H(ssk || yG || xG)$ is computed and sent to J_v . J_v verifies the message received from the client and if it is correct, J_v confirms that I knows the secret session key.

In the authentication phase, the random value w can be calculated on server side only if the servers know the long-lived symmetric keys, hence the client is able to verify the identity of multiple servers by checking h . On the other hand after calculating w on server side involving keys K_i , m_0 is checked. Correct m_0 proves that the client possesses K_i , hence identity of the client is verified as well. Value r_i ensures that the MAC value m_i is fresh for every authentication in order to avoid replay attack. Basically, the secret session key is created via an authenticated key agreement protocol based on random values (x, y) generated by the client and the selected server. These values are sent securely so the

attackers cannot gain any information about them. Considering the time complexity, the authentication phase is very efficient, since there is only one scalar multiplication on both sides besides the hash, MAC and xor operations.

4 Security Analysis

In this section after defining the security model, we provide a formal security proof of the proposed protocol. Basic requirements of the proposed protocol are mutual authentication of the participants, key secrecy, key freshness and key confirmation. Secure mutual authentication of participants prevents impersonation attack, the new key should be kept secret, and an old key shouldn't be exchanged successfully. At the end parties should confirm that the other party is able to use the new session key.

Known-key security and forward secrecy are also considered. If known-key security holds, disclosure of a session key does not jeopardize the security of other session keys. Forward secrecy holds if long-term secrets of one or more entities are compromised and the secrecy of previous session keys is not affected.

4.1 Security Model

We have extended the indistinguishability-based model proposed by M. Bellare and P. Rogaway in 1993 (see [3–5, 7]). The goal of applying multiple servers is to take into account the situation when the verifier server is corrupted, *i.e.* the long-lived keys and other login information stored in the server database are hacked. Multiple servers together provide secure user authentication, if at least one of the verifier servers is uncorrupted. We generalize the security model as follows.

If $\kappa \in \mathbb{N}$, then 1^κ denotes the string consisting of κ consecutive 1 bits. Let κ denote the security parameter. We fix a nonempty set ID of participants. ID is the union of the finite, disjoint, nonempty sets $Client = \{1, 2, \dots, T_1(\kappa)\}$ and $Server = \{1, 2, \dots, n = T_2(\kappa)\}$, where $T_i(\kappa)$, $i \in 1, 2$ is a polynomial bound on the number of participants in κ for some polynomial function T_i . Each participant is modelled by an oracle \prod_{I, J_v}^l , which simulates a participant I executing a protocol session in the belief that it is communicating with another participant J_v for the l th time, where $l \in \{1, \dots, T_3(\kappa)\}$ for some polynomial function T_3 . For each protocol run k servers are chosen. We assume that at least one of the k servers is not corrupted. Participant J_v chosen randomly out of the k servers conducts the protocol on server side, J_v communicates with I and the other $k - 1$ servers. Oracles keep transcripts, which contain all messages they have sent and received and the queries they have answered. Each participant $I \in Client$ holds long-lived symmetric keys exchanged with each server $J_i \in Server$, $i \in \{1, 2, \dots, n\}$ during registration.

4.2 Adversarial Model

The adversary \mathcal{A} is neither a client nor a server. \mathcal{A} is a probabilistic polynomial time Turing Machine with a query tape where oracle queries and their answers are written. \mathcal{A} is able to relay, modify, delay or delete messages. We assume that \mathcal{A} is allowed to make the following queries.

Send($\prod_{I,J}^i, M$): This oracle query models an active attack, allows \mathcal{A} to send the message M to oracle $\prod_{I,J}^i$, and the oracle returns a message (m) that the user instance sends in response to the message M . $\prod_{I,J}^i$ following the protocol steps also provides information whether the oracle is in state (δ) **Accepted**, **Rejected** or $*$. The query enables \mathcal{A} to initiate a protocol run between participants I and J by query **Send**($\prod_{I,J}^i, \lambda$). The oracle replies: m, δ .

Reveal($\prod_{I,J}^i$): This models an insecure usage of a session key. If oracle $\prod_{I,J}^i$ is in state **accepted**, holding a secret session key ssk , then this query returns ssk to \mathcal{A} . The oracle replies: ssk .

Corrupt($\prod_{I,J}, K'_{I,J}$): This oracle query models the corruption of a participant. Replying to this oracle query a participant oracle $\prod_{I,J}$ replies long-lived keys $K_{I,J}$ and I 's state, *i.e.* all the values stored by the participant I , moreover \mathcal{A} is allowed to replace the stored long-lived keys with any valid keys of \mathcal{A} 's choice $K'_{I,J}$. The oracle replies: $K_{I,J}, state_I$.

Test($\prod_{I,J}^i$): This oracle query models the semantic security of the secret session key. It is allowed to be asked only once in a protocol run. If participant I has **accepted** holding a secret session key ssk , then a coin b is flipped. If $b = 1$, then ssk is returned to the adversary, if $b = 0$, then a random value from the distribution of the session keys is returned.

We define \mathcal{A} 's advantage, the probability that \mathcal{A} can distinguish the session key held by the queried oracle from a random string, as follows:

$$Adv^{\mathcal{A}}(\kappa) = |Pr[\text{guess correct}] - 1/2|.$$

Participants' oracle instances are terminated when they finish a protocol run. They are in state **accepted**, if they decide to accept holding a secret session key denoted by ssk , after receipt of properly formulated messages. An oracle can be in state accepted before it is terminated. An oracle is **opened** or **corrupted**, if it has answered a query **Reveal**($\prod_{I,J}^i$) or **Corrupt**($\prod_{I,J}, K'_{I,J}$), respectively.

Moreover adversary \mathcal{A} is given access to $Mac(\cdot)$ and $Enc(\cdot)$ oracles as well.

The Threshold Hybrid Corruption Model. A model is a *strong corruption model* ([3]), if long-lived keys $K_{I,J}$ and all the values stored (e.g. randomly chosen secret values) by the participant I during the protocol run are transferred to \mathcal{A} . In case of the *weak corruption model* only the long-lived keys $K_{I,J}$ are transferred or replaced, the adversary does not completely compromise the machine. Other values generated and stored during the protocol run are not revealed.

Definition 6. We call a model threshold hybrid corruption model, if we assume that the client is uncorrupted and there are at least $n - k + 1$ uncorrupted servers out of the n servers, if k servers are chosen randomly for AKC. Moreover, the server chosen to communicate with the client is

1. uncorrupted, or
2. corrupted weakly and among the remaining servers there is at least one uncorrupted.

During the attack an experiment of running the protocol with an adversary \mathcal{A} is examined. After generating the keys and system parameters, \mathcal{A} initializes all participant oracles and asks polynomially number of oracle queries including $\text{Send}(\prod_{I,J}^i, M)$, $\text{Reveal}(\prod_{I,J}^i)$, $\text{Corrupt}(\prod_{I,J}, K'_{I,J})$ to the participant oracles. Finally \mathcal{A} asks a $\text{Test}(\prod_{I,J}^i)$ query.

In order to give the definition of a secure AKC protocol, we need to review the definition of conversation and matching conversation from [7]. They were also formalized in [4].

Definition 7. Consider an adversary \mathcal{A} and a participant oracle $\prod_{I,J}^s$. We define the conversation $C_{I,J}^s$ of $\prod_{I,J}^s$ as a sequence of

$$C_{I,J}^s = (\tau_1, \alpha_1, \beta_1), (\tau_2, \alpha_2, \beta_2), \dots, (\tau_m, \alpha_m, \beta_m),$$

where τ_i denotes the time when oracle query α_i and oracle reply β_i are given ($i = 1, \dots, m$).

Naturally $\tau_i > \tau_j$, iff $i > j$. \mathcal{A} terminates after receiving the reply β_m , i.e. does not ask more oracle queries. During a conversation the initiator and responder oracles are differentiated. $\prod_{I,J}^s$ is an initiator oracle if $\alpha_1 = \lambda$, otherwise it is a responder. Consider the definition for matching conversation when the number of protocol flows is odd.

Definition 8. Running protocol P in the presence of \mathcal{A} , we assume that the number of flows is $R = 2\rho - 1$, $\prod_{I,J}^s$ is an initiator and $\prod_{J,I}^t$ is a responder oracle that engage in conversations C and C' , respectively.

C' is a matching conversation to C , if there exist $\tau_0 < \tau_1 < \dots < \tau_{R-1}$ and $\alpha_1, \beta_1, \dots, \beta_{\rho-1}, \alpha_\rho$ such that C is prefixed by:

$$(\tau_0, \lambda, \alpha_1), (\tau_2, \beta_1, \alpha_2), \dots, (\tau_{2\rho-2}, \beta_{\rho-1}, \alpha_\rho),$$

and C' is prefixed by:

$$(\tau_1, \alpha_1, \beta_1), (\tau_3, \alpha_2, \beta_2), \dots, (\tau_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1}).$$

C is a matching conversation to C' , if there exist $\tau_0 < \tau_1 < \dots < \tau_R$ and $\alpha_1, \beta_1, \dots, \beta_{\rho-1}, \alpha_\rho$ such that C' is prefixed by:

$$(\tau_1, \alpha_1, \beta_1), (\tau_3, \alpha_2, \beta_2), \dots, (\tau_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1}), (\tau_{2\rho-1}, \alpha_\rho, *),$$

and C is prefixed by:

$$(\tau_0, \lambda, \alpha_1), (\tau_2, \beta_1, \alpha_2), \dots, (\tau_{2\rho-2}, \beta_{\rho-1}, \alpha_\rho).$$

If C is a matching conversation to C' and C' is a matching conversation to C , then $\prod_{I,J}^s$ and $\prod_{J,I}^s$ are said to have had matching conversation.

Matching conversation formalizes real-time communication between entities I and J , it is necessary to define authentication property of an AKC protocol. We give the definition of the event $\text{No-Matching}^A(\kappa)$ that is a modified version of the definition given in [7]. We leave out the requirement that $J \in \text{Server}$ is uncorrupted. In our multi-server setting each client communicates with a server that can be corrupted weakly, if there is at least one uncorrupted server from the k servers.

Definition 9. $\text{No-Matching}^A(\kappa)$ denotes an event when in a protocol P in the presence of an adversary \mathcal{A} assuming a threshold hybrid corruption model, there exist

1. a client oracle $\prod_{I,J}^s$ which is accepted, but there is no server oracle $\prod_{J,I}^t$ having a matching conversation with $\prod_{I,J}^s$, or
2. a server oracle $\prod_{I,J}^s$ which is uncorrupted and accepted, but there is no client oracle $\prod_{J,I}^t$ having a matching conversation with $\prod_{I,J}^s$, or
3. a server oracle $\prod_{I,J}^s$ which is corrupted weakly and accepted, but there is no client or no uncorrupted server oracle having a matching conversation with $\prod_{I,J}^s$.

In order to give the definition of a secure AKC, it is essential to define the notion of *freshness* and *benign adversary*.

Definition 10. A $k+1$ -tuple of oracles containing one client and k server oracles is *fresh*, if in the threshold hybrid corruption model the client oracle and the server oracle with which it has had a matching conversation are unopened. We call an oracle *fresh*, if it is an element of a fresh $k+1$ -tuple.

Definition 11. An adversary is called *benign* if it is deterministic, and restricts its action to choosing a $k+1$ tuple of oracles containing one client and k server oracles, and then faithfully conveying each flow from one oracle to the other, with the client oracle beginning first.

Definition 12. A protocol is a secure AKC protocol if,

1. In the presence of the benign adversary the client and the server oracle communicating with the client always accept holding the same session key ssk , and this key is distributed uniformly at random on $\{0, 1\}^\kappa$.

and if for every adversary \mathcal{A}

2. If in a threshold hybrid corruption model there is a server oracle $\prod_{I,J}^i$ having matching conversations with a client oracle and if $\prod_{I,J}^i$ is weakly corrupted, $\prod_{I,J}^i$ has matching conversation with an uncorrupted server oracle, then the client oracle and oracle $\prod_{I,J}^i$ both accept and hold the same session key ssk .

3. The probability of $\text{No-Matching}^{\mathcal{A}}(\kappa)$ is negligible.
 4. If the tested oracle is fresh, then $\text{Adv}^{\mathcal{A}}(\kappa)$ is negligible.

Theorem 1. *The proposed protocol is a secure AKC protocol in the random oracle model, assuming MAC is universally unforgeable under an adaptive chosen-message attack and symmetric encryption scheme is indistinguishable under chosen plaintext attack, moreover ECCDH assumption holds in the elliptic curve group.*

Proof. The conditions 1 and 2 hold, since the steps of the protocol are followed and with the assumption that the MAC and the encryption scheme provides correct verification and decryption, respectively. Moreover the hash function is a random oracle.

Let's look at condition 3. Consider an adversary \mathcal{A} and suppose that $\text{Pr}[\text{No-Matching}^{\mathcal{A}}(\kappa)]$ is non-negligible. There are two cases: either the server, or the client oracle is accepted.

– Case 1.

Let \mathcal{A} **succeeds** denote the event that in \mathcal{A} 's experiment there is a server oracle $\prod_{J_v, I}^t$ that is *accepted*, but there is no client oracle \prod_{I, J_v} having matching conversation to $\prod_{J_v, I}^t$.

We assume that

$$\text{Pr}[\mathcal{A} \text{ succeeds}] = n_S(\kappa),$$

where $n_S(\kappa)$ is non-negligible.

We construct a polynomial time adversary \mathcal{F} that is able to proceed an existential forgery against MAC under an adaptive chosen message attack. \mathcal{F} 's task is to generate a valid (m, t) message-tag pair, where m was never asked from the oracle $\text{Mac}_K(\cdot)$ for a security parameter κ . \mathcal{F} simulates the key generation Γ and answers \mathcal{A} ' oracle queries.

\mathcal{F} randomly picks $I \in \text{Client}$ and $J_1, \dots, J_k \in \text{Server}$, moreover randomly chooses $J_v \in \{J_1, \dots, J_k\}$ and $J_u \in \{J_1, \dots, J_k\}$. Let $\Delta = \{I, J_1, \dots, J_k\}$ denote identities of protocol participants. $\prod_{J_v, I}$ denotes the server oracle that communicates to the client I , \prod_{J_u, J_v} oracle denotes the uncorrupted server oracle that is in connection with server J_v . If $u = v$, then the server communicating with the client is uncorrupted. \mathcal{F} also chooses randomly a particular session $l \in \{1, \dots, T_3(\kappa)\}$. Given security parameter κ , adversary \mathcal{F} randomly chooses values K_1, \dots, K_k as long-lived keys exchanged between client I and servers J_1, \dots, J_k , and $\overline{K}_1, \dots, \overline{K}_{k-1}$ as encryption keys exchanged between J_v and J_1, \dots, J_{k-1} . \mathcal{F} runs \mathcal{A} and answers \mathcal{A} 's queries as follows.

1. \mathcal{F} answers H_0, H hash oracle queries at random (like a real random oracle would).
2. \mathcal{F} answers **Corrupt** queries according to Π , reveals long-lived keys K_i , internal states and encryption keys \overline{K}_i for corrupted servers. Queries to the uncorrupted server and the client oracles are refused. If $u \neq v$, then J_v is corrupted weakly, hence to the corrupt query \mathcal{F} answers only K_i and \overline{K}_i .

3. \mathcal{F} answers **Reveal** queries as specified in Π . This query is refused if it is asked from \prod_{I, J_v} or $\prod_{J_v, I}$.
4. \mathcal{F} answers **Send** queries according to Π with the knowledge of the keys, if they are not sent to $\prod_{J_v, I}$ and \prod_{J_u, J_v} . \mathcal{F} answers queries to \prod_{J_u, J_v} by choosing \bar{K}_u randomly. If \mathcal{A} does not involve $\prod_{J_v, I}$ as a server oracle which communicates to the client oracle \prod_{I, J_v} and other server oracles \prod_{J, J_i} , then \mathcal{F} gives up. If \mathcal{A} involves $\prod_{J_u, I}$ as an initiator oracle, then \mathcal{F} gives up. Otherwise \mathcal{A} asks query **Send**($\prod_{J_v, I}, M_1$), where

$$M_1 = I || J_1 || \dots || J_k || m_0 || \dots || m_k,$$

where $m_j = (Mac_{K_j}(r_j \oplus J_j) \oplus w_j) || r_j$ for corrupted servers, $r_j \in \{0, 1\}^\sigma$ are chosen randomly, $m_v = (Mac_{K_v}(r_v \oplus xG \oplus J_v) \oplus w_v) || r_v || xG$ for random $x \in \mathbb{Z}_n^*$ and m_u for uncorrupted J_u .

\mathcal{A} asks hash oracle queries $H(\cdot)$ to get w_i , \mathcal{F} answers these queries. If there is a J_t , $t = 1..k$ in M_1 such that $J_t \notin \Delta$, then \mathcal{F} gives up. \mathcal{A} asks MAC oracle queries to calculate m_u , \mathcal{F} answers these queries using his/her oracle $Mac_K(\cdot)$. Eventually \mathcal{A} creates a valid m_u . \mathcal{F} calculates the valid (\bar{m}, \bar{t}) MAC forgery, where $\bar{t} = MAC_K(\bar{m})$, as follows. If $u = v$, then $m_u = p || q || f$, $\bar{t} = p \oplus w_u$ and $\bar{m} = q \oplus f \oplus J_u$, where w_u is generated via oracle $H(\cdot)$. If $u \neq v$, then $m_u = p || q$, $\bar{t} = p \oplus w_u$, $\bar{m} = q \oplus J_u$. If \bar{m} was asked to oracle $Mac_K(\cdot)$ before, then \mathcal{F} gives up. \mathcal{F} answers query **Send**($\prod_{J_v, I}, M_1$) with $H(H_0(yxG) || yG || xG || w) || yG$, where $y \in \mathbb{Z}_n^*$ randomly chosen and w is calculated from w_i values of the corrupt servers and w_u . If some later time \mathcal{A} does not asks **Send**($\prod_{J_v, I}, H(H_0(xyG) || yG || xG)$), then \mathcal{F} gives up, otherwise $\prod_{J_v, I}$ gets accepted.

\mathcal{F} answers query **Send**(\prod_{I, J_v}, M_2), as follows. If

$M_2 \neq H(H_0(yxG) || yG || xG || w) || yG$, then \mathcal{F} gives up, otherwise replies $H(H_0(yxG) || yG || xG)$.

Finally, \mathcal{F} responses (\bar{m}, \bar{t}) to the challenger. If \mathcal{A} succeeds with non-negligible probability, then \mathcal{F} outputs a valid forgery (\bar{m}, \bar{t}) , where \bar{m} was never asked to oracle $Mac_K(\cdot)$ before.

Assume that \mathcal{A} is successful, event \mathcal{A} **succeeds** happens with $n_S(\kappa)$ non-negligible probability. Hence following the algorithm above \mathcal{F} calculates a valid (\bar{m}, \bar{t}) pair. We show that \mathcal{F} wins its experiment with non-negligible probability. The probability that \mathcal{F} chooses correct participants Δ , session l and succeeds is

$$\xi_1(\kappa) = \frac{n_S(\kappa)}{T_1(\kappa)T_2(\kappa)\binom{T_2(\kappa)-1}{k-1}T_3(\kappa)} - \lambda(\kappa), \quad (1)$$

where $\lambda(\kappa)$ denotes the probability that \mathcal{F} previously calculated the flow. Since $n_S(\kappa)$ is non-negligible, $T_i(\kappa)$ ($i=1, \dots, 3$) is polynomial in κ and $\lambda(\kappa)$ is negligible and

$$\xi_1(\kappa) \geq \frac{n_S(\kappa)}{T_1(\kappa)T_2(\kappa)T_2(\kappa)^{k-1}T_3(\kappa)} - \lambda(\kappa),$$

thus $\xi_1(\kappa)$ is non-negligible. That contradicts the security assumption of MAC, hence $n_S(\kappa)$ must be negligible.

Further details of the proof can be found in Appendix A.

The proposed AKC scheme possesses known-key and forward secrecy. Key parameters xG and yG for each run are chosen independently and randomly, hence session keys are also independent and random. The adversary is able to calculate the session key from xG and yG only if he or she computes ECCDH function.

If the adversary asks $\text{Reveal}(\prod_{I,J}^i)$ and receives a session key, the session keys of the subsequent runs are independent from the revealed session key, hence the scheme is known-key secure with the ECCDH assumption.

In the proposed AKC values x, y are not transmitted for key parameters xG and yG , hence if long-term secret keys are compromised, the adversary faces ECCDH assumption for the previous session keys.

4.3 Practical Issues

Efficiency was an important aspect during the design of our protocol. In the protocol, the session key is generated by ECDH key exchange, and the other operations are hash and xor operations, which are extremely fast. After the security analysis, a java application was created to simulate the protocol in a real environment. The system is divided into two applications, the first one is the client, and the other one is the application for implementing the servers. Both applications were run in a microsoft azure cloud environment for authentic simulation. We worked with constant parameters in the simulation, because the difference in user names and passwords was negligible in terms of performance. Clients and the selected server communicate on the public channel. The communication between the servers occurs through a encrypted channel. In connection with their operation, it should be noted that the server programs monitor the incoming authentication requests in the background. Furthermore, when running the client program, the user selects the servers that will perform the authentication. After calculating the first message, the user sends it to the server that communicates with the service provider and communicates with the other servers. During the connection, the steps described in the protocol are implemented. If the connection fails, the server rejects the connection request and keeps track of the channel.

Java Simulation Result. Java simulation was tested with six servers, one of which received the client authentication request and the remaining five servers were required for authentication. Time means the time interval from the first step of the protocol to the successful connection. The average connection time was then calculated based on the received time results. For the cloud environment, we tested the standard package and the p3v2 package. The Standard Package

contains 100 Azure compute units [27] and the p3v2 package includes 840 Azure compute units. The concept of the Azure Compute Unit (ACU) provides a way of comparing compute (CPU) performance across Azure Stock Keeping Units (SKUs). This will help you to identify easily the SKU which is the most likely to satisfy your performance needs. ACU is currently standardized on a Small (Standard_A1) VM being 100 and all other SKUs then represent approximately how much faster that SKU can run a standard benchmark.

The results are the following: The average connection time is 0,1092 s per connection when we use the p3v2 package. In the standard package the result of average connection time is 0,1537 s per connection. This time contains the mutual authentication including the authentication of selected servers and generating a session key (Fig. 2).

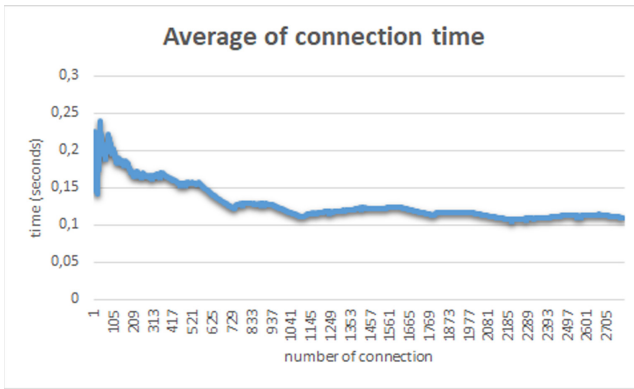


Fig. 2. Average connection times of the successful authentications between the client and the servers.

It is an important issue in our protocol that the user gives his/her password and after that the long-lived keys are available. The static password is comfortable for the user and the long-lived keys provide the appropriate security level. Since in each authentication the values are random and fresh, the key freshness holds and the protocol execution could not be successfully finished with old, already used values and keys.

5 Conclusion

Instead of the centralized authentication we have designed an Authenticated Key Agreement with Key Confirmation protocol for a distributed environment. Our system is robust against server breakdown and applies multiple-server identity verification. We introduce the threshold hybrid corruption model and we model when the different servers become corrupt on the protocol. We give a detailed security analysis, and we prove that our proposed protocol is a secure AKC

protocol in the random oracle model, assuming MAC is universally unforgeable under an adaptive chosen-message attack and symmetric encryption scheme is indistinguishable under chosen plaintext attack, moreover ECCDH assumption holds in the elliptic curve group. It is also important to note that during the authentication phase fast ECDH key exchange, MAC, xor operations and symmetric encryption are used, hence we achieve good results in computational time.

Acknowledgement. We thank the reviewers for their valuable comments which helped to improve the proposed protocol.

Appendix A

Proof. – Case 2.

Let \mathcal{A} **succeeds** denote the event that in \mathcal{A} 's experiment there is a client oracle \prod_{I,J_v}^s that is *accepted*, but there is no server oracle $\prod_{J_v,I}$ having matching conversation to \prod_{I,J_v}^s . We assume that

$$Pr[\mathcal{A} \text{ succeeds}] = n_C(\kappa),$$

where $n_C(\kappa)$ is non-negligible.

We can construct a polynomial time adversary that is able to distinguish two plaintexts under chosen plaintext attack against the symmetric encryption scheme.

Challenger generates a key K and flips a bit b . \mathcal{F} is given an oracle access to $Enc_K(\cdot)$. \mathcal{F} 's task is to output a bit b' on inputs m_0, m_1 chosen by \mathcal{F} and m_b . \mathcal{F} picks the protocol participants and a session $l \in \{1, \dots, T_3(\kappa)\}$, let $\Delta = \{I, J_1, \dots, J_k\}$ denote the identities. Similarly to Case 1, \prod_{I,J_v} denotes the client, $\prod_{J_v,I}$ the communicating server and \prod_{J_u,J_v} the uncorrupted server oracle. If $u = v$, then the server communicating with the client is uncorrupted. \mathcal{F} simulates the key generation Γ in the same way as in Case 1. \mathcal{F} generates long-lived keys K_i and symmetric encryption keys \bar{K}_i for corrupted servers for the security parameter κ . \mathcal{F} answers \mathcal{A} 's oracle queries as follows.

\mathcal{F} answers queries to oracles $H(\cdot), H_0(\cdot), \text{Corrupt}, \text{Reveal}$ in the same way as in Case 1. \mathcal{F} answers **Send** queries according to Π with the knowledge of the keys of corrupted servers, if they are not sent to $\prod_{I,J_v}, \prod_{J_v,I}$ or \prod_{J_u,J_v} . If \mathcal{A} does not involve $\prod_{J_v,I}$ as a server oracle which communicates to the client oracle \prod_{I,J_v} and other server oracles \prod_{I,J_i} , then \mathcal{F} gives up. We consider the case when $\prod_{J_v,I}$ is weakly corrupted ($u \neq v$). If \mathcal{A} does not invoke \prod_{I,J_v} as an initiator oracle, then \mathcal{F} gives up, otherwise \mathcal{A} asks oracle query **Send**(\prod_{I,J_v}^s, λ). \mathcal{F} responses $M_1 = I||J_1||\dots||J_k||m_0||\dots||m_k$, with $m_j = (Mac_{K_j}(r_j \oplus J_j) \oplus w_j)||r_j$ and $m_v = (Mac_{K_v}(r_v \oplus xG \oplus J_v) \oplus w_v)||r_v||xG$, where $r_i \in \{0, 1\}^\sigma, x \in \mathbb{Z}_n^*, w_i$ and the MAC key K_u for the uncorrupted server are randomly chosen by \mathcal{F} . Some time later \mathcal{A} asks oracle queries to $Enc(\cdot)$ and eventually asks query **Send**($\prod_{J_u,J_v}, I||m_u$).

\mathcal{F} answers with $Enc_K(m_b)$. \mathcal{F} perfectly simulates uncorrupted server \prod_{J_u,J_v} to \mathcal{A} , since without the knowledge of the MAC key \mathcal{A} cannot verify correctness

of m_u . \mathcal{A} some time later asks xyG to oracle $H_0(\cdot)$ and $H_0(xyG)||yG||xG||w$ to oracle $H(\cdot)$ and asks query $\text{Send}(\prod_{I,J_v}, H(H_0(xyG)||yG||xG||w)||yG)$. \mathcal{F} answers queries, replies $H(H_0(xyG)||yG||xG)$, gets accepted and checks whether $w = H(w_1||\dots||m_0||\dots||w_{k-1})$, where w_1, \dots, w_{k-1} denote the random values generated for corrupted servers. If the equality holds, then \mathcal{F} outputs bit $b' = 0$, otherwise $b' = 1$.

Assume that \mathcal{A} is successful, event \mathcal{A} **succeeds** happens with $n_C(\kappa)$ non-negligible probability. \mathcal{F} outputs the correct b' . We show that \mathcal{F} wins its experiment with non-negligible probability. For the analysis the probability that \mathcal{F} chooses the correct participants Δ , session l and succeeds is calculated:

$$\xi_2(\kappa) = \frac{n_C(\kappa)}{T_1(\kappa)T_2(\kappa)\binom{T_2(\kappa)-1}{k-1}T_3(\kappa)} - \lambda(\kappa), \quad (2)$$

where $\lambda(\kappa)$ denotes the probability that \mathcal{F} previously calculated the flow, including the case of uncorrupted $\prod_{J_v, I}$, when \mathcal{F} calculates correct MAC message-tag pair for $\prod_{J_v, I}$. Similarly to Case 1. $\xi_2(\kappa)$ is non-negligible, if $n_C(\kappa)$ is non-negligible, $T_i(\kappa)$ ($i=1, \dots, 3$) is polynomial in κ and $\lambda(\kappa)$ is negligible. That contradicts the security assumption of the symmetric encryption, hence $n_C(\kappa)$ must be negligible.

We turn to condition 4. Consider an adversary \mathcal{A} and suppose that $Adv^{\mathcal{A}}(\kappa)$ is non-negligible.

– Case 3.

Let \mathcal{A} **succeeds** against $\prod_{I,J}^s$ denote the event that \mathcal{A} asks $\text{Test}(\prod_{I,J}^s)$ query and outputs the correct bit. Hence

$$Pr[\mathcal{A} \text{ succeeds}] = \frac{1}{2} + n(\kappa),$$

where $n(\kappa)$ is non-negligible.

Let A_κ denote the event that \mathcal{A} picks either a server or a client oracle $\prod_{I,J}^s$ and asks its Test query such that oracle $\prod_{I,J}^s$ has had a matching conversation to $\prod_{J,I}^t$.

$$Pr[\mathcal{A} \text{ succeeds}] = Pr[\mathcal{A} \text{ succeeds}|A_\kappa]Pr[A_\kappa] +$$

$$Pr[\mathcal{A} \text{ succeeds}|\bar{A}_\kappa]Pr[\bar{A}_\kappa]$$

According to the previous section $Pr[\bar{A}_\kappa] = \mu(\kappa)$, where $\mu(\kappa) \in \{n_C(\kappa), n_S(\kappa)\}$ and $Pr[A_\kappa] = 1 - \mu(\kappa)$, where $\mu(\kappa)$ is negligible, hence

$$\frac{1}{2} + n(\kappa) \leq Pr[\mathcal{A} \text{ succeeds}|A_\kappa]Pr[A_\kappa] + \mu(\kappa)$$

and we get

$$\frac{1}{2} + n_1(\kappa) = Pr[\mathcal{A} \text{ succeeds}|A_\kappa],$$

for a non-negligible $n_1(\kappa)$. Let B_κ denote the event that for given aG, bG adversary \mathcal{A} or any other oracle besides $\prod_{I,J}^s$ or $\prod_{J,I}^t$ asks abG to oracle $H_0(\cdot)$. $Pr[\mathcal{A} \text{ succeeds}|A_\kappa] = Pr[\mathcal{A} \text{ succeeds}|A_\kappa \wedge B_\kappa]Pr[B_\kappa|A_\kappa] + Pr[\mathcal{A} \text{ succeeds}|A_\kappa \wedge \bar{B}_\kappa]Pr[\bar{B}_\kappa|A_\kappa]$. Since $Pr[\mathcal{A} \text{ succeeds}|A_\kappa \wedge \bar{B}_\kappa] = \frac{1}{2}$,

$$\frac{1}{2} + n_1(\kappa) \leq Pr[\mathcal{A} \text{ succeeds}|A_\kappa \wedge B_\kappa]Pr[B_\kappa|A_\kappa] + \frac{1}{2}, \tag{3}$$

hence $Pr[B_\kappa|A_\kappa]$ is non-negligible.

We construct a polynomial time adversary \mathcal{F} that for given aG, bG calculates $ECCDH(aG, bG) = abG$. \mathcal{F} picks the protocol participants, $\Delta = \{I, J_1, \dots, J_k\}$ denotes the identities. Similarly to Case 1, \prod_{I,J_v} denotes the client, $\prod_{J_v,I}$ the communicating server and \prod_{J_u,J_v} the uncorrupted server oracle. If $u = v$, then the server communicating with the client is uncorrupted. \mathcal{F} sets $par = (E, q, \mathbf{n}, G, H, H_0, Mac)$ public parameters, where $G \in E(\mathbb{F}_q)$ is a generator of order \mathbf{n} . \mathcal{F} also simulates the key generation Γ in the same way as in Case 1.

\mathcal{F} answers queries to oracles $H(\cdot), H_0(\cdot), \text{Corrupt}, \text{Reveal}$ in the same way as in Case 1. Let $T_4(\kappa)$ denote the polynomial bound on the number of queries allowed to ask to oracle $H_0(\cdot)$. \mathcal{F} randomly picks $j \in \{1, \dots, T_4(\kappa)\}$, assuming that j th query will be on abG . \mathcal{F} answers **Send** queries according to Π except for queries to \prod_{I,J_v}^s and $\prod_{J_v,I}^t$. \mathcal{F} generates messages to \prod_{I,J_v}^s and $\prod_{J_v,I}^t$ in a way that instead of choosing x, y randomly inserts aG, bG as inputs. The MAC computations of message M_1 are calculated by the randomly chosen keys K_i and aG or bG is inserted. M_2 sent to \prod_{I,J_v}^s is constructed as a concatenation of h and bG or aG , respectively, where h is a freshly generated random value. If \mathcal{A} does not ask j queries to $H_0(\cdot)$, then \mathcal{F} gives up. After the j th query \mathcal{F} stops and outputs the j th query. If \prod_{I,J_v}^s and $\prod_{J_v,I}^t$ do not have matching conversation, then \mathcal{F} gives up.

The probability that \mathcal{F} succeeds is at least

$$\xi_3(\kappa) = \frac{n_1(\kappa)}{T_1(\kappa)T_2(\kappa)\binom{T_2(\kappa)-1}{k-1}T_3(\kappa)^2T_4(\kappa)} - \mu(\kappa),$$

that is non-negligible. This contradicts to the ECCDH assumption, hence $n_1(\kappa)$ and $Adv^{\mathcal{A}}(\kappa)$ must be negligible.

References

1. Acar, T., Belenkiy, M., K upc u, A.: Single password authentication. *Comput. Netw.* **57**(13), 2597–2614 (2013)
2. Bagherzandi, A., Jarecki, S., Saxena, N., Lu, Y.: Password-protected secret sharing. In: ACM Conference on Computer and Communications Security (2011)
3. 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>

4. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_21
5. Bellare, M., Rogaway, P.: Provably secure session key distribution: the three party case. In: Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, pp. 57–66 (1995)
6. Bellare, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy. IEEE (1992)
7. Blake-Wilson, S., Johnson, D., Menezes, A.: Key agreement protocols and their security analysis. In: Darnell, M. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 30–45. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0024447>
8. Boyko, V., MacKenzie, P., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_12
9. Boyen, X.: Hidden credential retrieval from a reusable password. In: Proceedings of the 4th International Symposium on Information, pp. 228–238. ACM (2009)
10. Boyen, X.: HPAKE: password authentication secure against cross-site user impersonation. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 279–298. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10433-6_19
11. Brainard, J., Juels, A., Kaliski, B., Szydło, M.: A new two-server approach for authentication with short secrets. In: Proceeding SSYM 2003, Proceedings of the 12th Conference on USENIX Security Symposium, vol. 12, pp. 1–14 (2003)
12. Chen, N., Jiang, R.: Security analysis and improvement of user authentication framework for cloud computing. *J. Netw.* **9**(1), 198–203 (2014)
13. Choudhury, A.J., Kumar, P., Sain, M.: A strong user authentication framework for cloud computing. In: Proceedings of IEEE Asia-Pacific Services Computing Conference, pp. 110–115 (2011)
14. Ford, W., Kaliski, B.S.: Server-assisted generation of a strong secret from a password. In: Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 2000. IEEE (2000)
15. Hassanzadeh-Nazarabadi, Y., Küpçü, A., Özkasap, O.: LightChain: a DHT-based blockchain for resource constrained environments. arXiv preprint [arXiv:1904.00375](https://arxiv.org/abs/1904.00375) (2019)
16. Huszti, A., Oláh, N.: A simple authentication scheme for clouds. In: Proceedings of IEEE Conference on Communications and Network Security (CNS), pp. 565–569 (2016)
17. Hwang, M.S., Li, L.H.: A new remote user authentication scheme using smart cards. *IEEE Trans. Consum. Electron.* **46**(1), 28–30 (2000)
18. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 233–253. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_13
19. İşler, D., Küpçü, A.: Threshold single password authentication. In: Garcia-Alfaro, J., Navarro-Arribas, G., Hartenstein, H., Herrera-Joancomartí, J. (eds.) ESORICS/DPM/CBT -2017. LNCS, vol. 10436, pp. 143–162. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67816-0_9
20. İşler, D., Küpçü, A.: Distributed Single Password Protocol Framework. *IACR Cryptol. ePrint Arch.*, p. 976 (2018)

21. Katz, J., MacKenzie, P., Taban, G., Gligor, V.: Two-server password-only authenticated key exchange. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 1–16. Springer, Heidelberg (2005). https://doi.org/10.1007/11496137_1
22. Katz, J., Ostrovsky, R., Yung, M.: Efficient password-authenticated key exchange using human-memorable passwords. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 475–494. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_29
23. Ku, W.C., Chen, S.M.: Weaknesses and improvements of an efficient password based remote user authentication scheme using smart cards. *IEEE Trans. Consum. Electron.* **50**(1), 204–207 (2004)
24. Kurtz, G., Alperovitch, D., Zaitsev, E.: Hacking Exposed: Beyond the Malware, RSA 2015 (slide deck) (2015). https://www.rsaconference.com/writable/presentations/file_upload/expt10_hackingexposedbeyondthemalware.pdf
25. MacKenzie, P., Shrimpton, T., Jakobsson, M.: Threshold password-authenticated key exchange. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 385–400. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_25
26. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_32
27. <https://docs.microsoft.com/en-us/azure/virtual-machines/acu>
28. Soria-Machado, M., Abolins, D., Boldea, C., Socha, K.: Kerberos Golden Ticket Protection, Mitigating Pass-the-Ticket on Active Directory, CERT-EU Security Whitepaper 2014-007 (2016)
29. <https://www.entrepreneur.com/article/295831>
30. <https://www.openstack.org/>
31. Di Raimondo, M., Gennaro, R.: Provably secure threshold password-authenticated key exchange. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 507–523. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_32
32. Saeed, M.E.S., Liu, Q.-Y., Tian, G.Y., Gao, B., Li, F.: AKAIoTs: authenticated key agreement for Internet of Things. *Wirel. Netw.* **25**(6), 3081–3101 (2018). <https://doi.org/10.1007/s11276-018-1704-5>
33. Sood, S.K., Sarje, A.K., Singh, K.: A secure dynamic identity based authentication protocol for multi-server architecture. *J. Netw. Comput. Appl.* **34**(2), 609–618 (2011)
34. Wu, T., Lee, Z., Obaidat, M.S., Kumari, S., Kumar, S., Chen, C.: An authenticated key exchange protocol for multi-server architecture in 5G networks. *IEEE Access* **8**, 28096–28108 (2020). <https://doi.org/10.1109/ACCESS.2020.2969986>



Forward-Secure 0-RTT Goes Live: Implementation and Performance Analysis in QUIC

Fynn Dallmeier², Jan P. Drees¹, Kai Gellert¹(✉), Tobias Handirk¹,
Tibor Jager¹, Jonas Klauke², Simon Nachtigall², Timo Renzelmann²,
and Rudi Wolf²

¹ University of Wuppertal, Wuppertal, Germany
{jan.drees,kai.gellert,tobias.handirk,tibor.jager}@uni-wuppertal.de
² Paderborn University, Paderborn, Germany

Abstract. Modern cryptographic protocols, such as TLS 1.3 and QUIC, can send cryptographically protected data in “zero round-trip times (0-RTT)”, that is, without the need for a prior interactive handshake. Such protocols meet the demand for communication with minimal latency, but those currently deployed in practice achieve only rather weak security properties, as they may not achieve forward security for the first transmitted payload message and require additional countermeasures against replay attacks.

Recently, 0-RTT protocols with full forward security and replay resilience have been proposed in the academic literature. These are based on *puncturable encryption*, which uses rather heavy building blocks, such as cryptographic pairings. Some constructions were claimed to have practical efficiency, but it is unclear how they compare concretely to protocols deployed in practice, and we currently do not have any benchmark results that new protocols can be compared with.

We provide the first concrete performance analysis of a modern 0-RTT protocol with full forward security, by integrating the Bloom Filter Encryption scheme of Derler *et al.* (EUROCRYPT 2018) in the Chromium QUIC implementation and comparing it to Google’s original QUIC protocol. We find that for reasonable deployment parameters, the server CPU load increases approximately by a factor of eight and the memory consumption on the server increases significantly, but stays below 400 MB even for medium-scale deployments that handle up to 50K connections per day. The difference of the size of handshake messages is small enough that transmission time on the network is identical, and therefore not significant.

We conclude that while current 0-RTT protocols with full forward security come with significant computational overhead, their use in practice is feasible, and may be used in applications where the increased CPU

Supported by the German Research Foundation (DFG), project JA 2445/2-1 and the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme, grant agreement 802823. Part of this work was completed while the authors were employed at Paderborn University.

and memory load can be tolerated in exchange for full forward security and replay resilience on the cryptographic protocol level. Our results serve as a first benchmark that can be used to assess the efficiency of 0-RTT protocols potentially developed in the future.

1 Introduction

0-RTT Protocols. Considerable effort has gone into reducing latency at the various networking layers, with the aim of reducing end-to-end latencies. This includes HTTP/2 [5] based on Google’s SPDY protocol [6], TCP Fast Open [12] and μ TP [28], as well as the move towards decentralized content delivery networks or peer-to-peer communication. Still, classical cryptographic key agreement protocols, such as TLS 1.2, require at least one round-trip time (RTT) to establish a key, plus an additional RTT to establish the underlying TCP session. In order to overcome this, 0-RTT protocols have been developed and incorporated in transport layer standards. This is motivated by improved user experience, which degrades with increased latency between user actions (such as requesting a web page) and the result (the web page being displayed), as well as the demand for fast session establishment in applications with extreme latency requirements, such as real-time control of industrial systems over 5G networks, for instance. Studies showed that for large Internet companies, such as Google or Amazon, additional delays have tangible impact on revenue [9, 23]. Google has approached this latency issue with their QUIC protocol [11], which includes a 0-RTT key exchange protocol.

Security of 0-RTT Protocols. Fundamentally, a key exchange mechanism is designed to establish a common authenticated secret key between communication partners. One classical security requirement is *replay resilience*, which essentially means that an adversary should not be able to replay cryptographically protected messages in a way that tricks the receiver into processing the replayed message again, which in turn may cause a duplicated execution of a command, for example.

Another fundamental requirement is *forward security*, which is a standard security property expected from modern cryptographic protocols. Consider a case where a server is compromised and secret key material is leaked. Forward security essentially means that an adversary that has recorded previous sessions is not able to retroactively decrypt the data exchanged earlier. Hence, forward security ensures that the disclosure of a secret key does not compromise the confidentiality of earlier communication.

Both replay resilience and forward security are difficult to achieve with 0-RTT key exchange protocols. The fundamental challenge is the missing interactivity between the client and the server, as the client needs to be able to encrypt data without getting any new information (e.g., a Diffie–Hellman share with fresh randomness) from the server. For a comprehensive discussion on forward security in non-interactive settings, we refer the reader to [8].

The QUIC Protocol. When considering latency in the networking stack, it is apparent that the minimization of the overall number of necessary round-trips must consider multiple layers. Google has introduced SPDY, which has been standardized as HTTP/2 since, to address the application layer on the world wide web. It remains to consider the transport layer connection establishment and the cryptographic handshake. Traditionally, each of these would take at least 1 RTT for the handshake process. The QUIC protocol was developed by Google to address both at once. In order to avoid the latency incurred by a TCP handshake, QUIC is based on the lighter UDP protocol. This makes it necessary for QUIC to define additional protocol operations on top of UDP to retain some of the guarantees needed for reliable operation of higher layers, such as improved congestion control, multiplexing without head-of-line blocking, forward error correction and connection migration [27].

QUIC implements a custom cryptographic protocol, based on the Diffie-Hellman key exchange, see Sect. 2.1 for a detailed protocol description. Essentially, the very first connection of a client to a server over QUIC still requires a 1-RTT cryptographic handshake. During this handshake, a SERVER CONFIGURATION message is sent from the server to the client. This message contains a medium-lived (typically two days) Diffie-Hellman share g^s of the server, which is digitally signed with the server’s long-term signature key, as well as information about supported cryptographic algorithms and their parameters.

On subsequent connections the SERVER CONFIGURATION data can then be used to perform a 0-RTT key exchange. To this end, the client selects supported parameters and a Diffie-Hellman share g^x , which yields an initial session key g^{xs} that can then be used immediately to encrypt the payload data m sent from the client to the server as $c = \text{Enc}(g^{xs}, m)$. Note that the server re-uses the same ephemeral randomness g^s for all sessions within the lifetime of the SERVER CONFIGURATION message. Hence, an attacker that obtains s is able to compute the key of all sessions within this time period. Therefore only a weak form of forward security is achieved, which holds only *after* the corresponding SERVER CONFIGURATION message has expired [24].

Furthermore, it is well-known that neither Google’s QUIC [11], nor the protocol version standardized by the IETF [21], protect against replay attacks. An attacker can replay a 0-RTT message $(g^x, \text{Enc}(g^{xs}, m))$ to the server over and over again. Without additional application-layer countermeasures, this would trick the server into repeatedly processing the payload message m .

Puncturable Key Encapsulation. 0-RTT protocols with full forward security and replay resilience therefore follow a different approach than TLS 1.3 and QUIC, by using *puncturable key encapsulation*. This approach was introduced in [20], following [19], and is also used in the more efficient Bloom Filter key encapsulation schemes from [14, 15] considered in this paper. A 0-RTT protocol essentially consists of a *puncturable* key encapsulation mechanism (KEM), which is used to transport a random session key from the client to the server. After the server receives the ciphertext C encapsulating the session key, it decrypts the ciphertext with the corresponding secret key, and then immediately “punctures” the secret

key. A punctured key cannot be used to decrypt C . Even given the punctured key, the key encapsulated in C is indistinguishable from random. It is furthermore possible to repeatedly puncture a secret key with respect to different ciphertexts, which makes the 0-RTT protocol usable for multiple sessions.

Our Contributions. We implemented the Bloom Filter key encapsulation mechanism based on identity-based broadcast encryption (IBBE) introduced in [15], instantiated with the IBBE scheme by Delerablée [13]. We optimized the implementation with regard to speed, utilizing parallelization and pre-computations where possible, and integrated the scheme into the QUIC protocol implementation of Chromium [2]. Our repository can be found at <https://gitlab.com/buw-itsc/fs0rtt>.

We analyzed the computational performance of the new protocol, comparing it to Google’s implementation of the QUIC key exchange in Chromium. Specifically, we measured server and client memory consumption, handshake duration, size of the exchanged messages, maximum server throughput and server CPU load. This yields the first benchmark result for recent 0-RTT protocols with full forward security and replay resilience.

Results. We find that for reasonable deployment parameters (and despite the use of computationally heavy building blocks, such as pairings), the server CPU load increases approximately by a factor of eight, while the number of handshakes the server is able to process per second is reduced by the same factor. The memory consumption on the server increases significantly, but stays below 400 MB even for medium-scale deployments that handle up to 50K connections per day. The size of the first 1-RTT handshake increases by 18% and the following 0-RTT handshakes decreases by 13% when compared to the QUIC protocol. The increase of the first message is small enough that transmission time on the network is identical, and therefore not significant. As to handshake duration, the 0-RTT (resp. 1-RTT) handshake takes approximately eight times (resp. 1.6 times) longer in comparison to the respective QUIC handshakes.

The increased computation time of the 0-RTT handshake is still lower than that of a full 1-RTT handshake in the QUIC protocol. This means that even when the improvements provided by the reduced number of round trips, which will vary depending on network speed and latency, is not taken into account, the forward-secure 0-RTT mode is still preferable over 1-RTT from a latency perspective.

Our Choice of Protocols. The only current real-world implementations of 0-RTT protocols are QUIC, implemented in the Chromium browser and the web server implementations of LiteSpeed and Nginx¹ as well as the 0-RTT mode of TLS 1.3. QUIC runs on top of the UDP transport layer protocol, which does not require a handshake and therefore truly achieves 0-RTT session establishment. However, UDP provides only an unreliable best-effort channel, therefore QUIC

¹ See https://w3techs.com/technologies/segmentation/ce-quic/web_server.

additionally implements transport-layer algorithms that deal with package loss, perform package re-transmission, and implement congestion control. In contrast, TLS 1.3 runs on top of the TCP protocol, which provides a reliable channel and congestion control, but requires an initial handshake and therefore adds another RTT latency.

Our objective is to provide the first benchmark of the *real-world* performance of a forward-secure 0-RTT protocol. Therefore it makes sense to consider a protocol implementation that runs on top of UDP, as otherwise the latency incurred by the TCP handshake would blur the measurements and yield less clear results. Furthermore, we want to consider a real-world setting where algorithms to deal with packet loss in UDP are implemented, but we want to avoid that the particular choice of these algorithms or the performance of their implementation impacts our measurements. Hence, in order to obtain an as-meaningful-as-possible comparison, our new implementation should use exactly the same transport layer protocol stack and additional transport layer algorithms as the protocol we compare with.

Therefore we chose to base our implementation on QUIC, where we replace only the cryptographic core with a forward-secure 0-RTT protocol, but re-use all other functionality without any modification. This provides the most clear results and the most objective comparison of the performance impact of the modified cryptographic core of the protocol.

Furthermore, we chose to use a Bloom Filter KEM as the basis, as it allows for significantly more efficient puncturing (by several orders of magnitude) than the tree-based constructions from [19, 20]. While one could ask for a comparison to other Bloom Filter KEMs, we claim that these will yield less efficient protocols and thus are out of scope of our work. Our objective is not to compare the performance of different (theoretical) 0-RTT protocol instantiations, but to assess how a modern forward-secure 0-RTT protocol compares to the protocols currently used in practice.

Related Work. To our best knowledge, this is the first work that experimentally assesses the computational performance and resource requirements of 0-RTT protocols with full forward security. The QUIC protocol was introduced in [27] and formally analyzed by Lychev *et al.* [24]. The idea of puncturable encryption was introduced in [19], the idea of using it to construct fully forward-secure 0-RTT protocols in [20]. Bloom Filter Encryption was introduced as a more efficient variant in [14, 15]. Lauer *et al.* [22] used Bloom filter encryption to construct a single-pass circuit construction protocol with full forward security, which resembles a multi-hop 0-RTT protocol. However, it was not implemented and, to the best of our knowledge, no experimental performance assessments of 0-RTT-like protocols with full forward security have been made so far. Aviram *et al.* [4] have developed techniques to overcome the lack of forward security and replay resilience in 0-RTT session resumption protocols, such as the 0-RTT mode in TLS 1.3. Their techniques allow an efficient solution to this problem by utilizing only private-key primitives. These techniques, however, consider a different setting, which is based on a shared symmetric key between a client

and a server, and requires secure storage on the client. In contrast, we consider “real” 0-RTT protocols where only public information (the server’s public key) is stored on the client.

We remark that, similar to our approach, the UDP-based transport layer of QUIC is currently in the process of being standardized with TLS 1.3 as cryptographic core [29] (replacing the original QUIC key exchange protocol). However, note that TLS 1.3 only deploys a 0-RTT session resumption protocol that relies on key material, which has been established in a previous session. This 0-RTT mode is therefore incomparable to our 0-RTT key exchange where we only rely on publicly available information.

2 Protocol Design

In the following, we summarize the basic functionality of the handshake protocol in QUIC and explain why it does not provide forward security and replay resilience for 0-RTT data. Then we introduce Bloom filter encryption and discuss our parameter choice. Finally, we outline the handshake we implemented.

2.1 QUIC Handshake Protocol

The QUIC protocol uses symmetric encryption to ensure the confidentiality of the data exchanged between client and server. The necessary session key is derived using a modified Diffie–Hellman (DH) key exchange. Figure 1 shows the message flow for this key exchange.

Upon the start of a server, a `SERVER CONFIGURATION` is generated. This `SERVER CONFIGURATION` contains a DH share g^s with a freshly sampled exponent s and an expiration date. A fresh `SERVER CONFIGURATION` is generated periodically, typically every two days.

Initially, a client does not possess any information about the `SERVER CONFIGURATION`. Therefore, it initiates a 1-RTT connection using an `INCHOATE CLIENT HELLO`. The server responds with its `SERVER CONFIGURATION` and a signature on the `SERVER CONFIGURATION` in a `REJECTION`. The client stores the `SERVER CONFIGURATION` for upcoming 0-RTT connections if the signature is valid. Note that the same `SERVER CONFIGURATION` of a server is shared among *all* clients during its lifetime. In case the client uses an out-of-date `SERVER CONFIGURATION`, it reinitiates a 1-RTT connection by sending an `INCHOATE CLIENT HELLO` to receive a new one.

If the client is in possession of a `SERVER CONFIGURATION`, it initiates a 0-RTT connection. To this end, the client establishes an initial key g^{sx} using the DH share g^s contained in the `SERVER CONFIGURATION` and its own freshly sampled exponent x . The initial key is used to encrypt and send 0-RTT data alongside with the client’s DH share g^x to the server in a `CLIENT HELLO`. When the server extracts the client’s DH share from the `CLIENT HELLO`, it can also compute the initial key to decrypt the encrypted 0-RTT data.

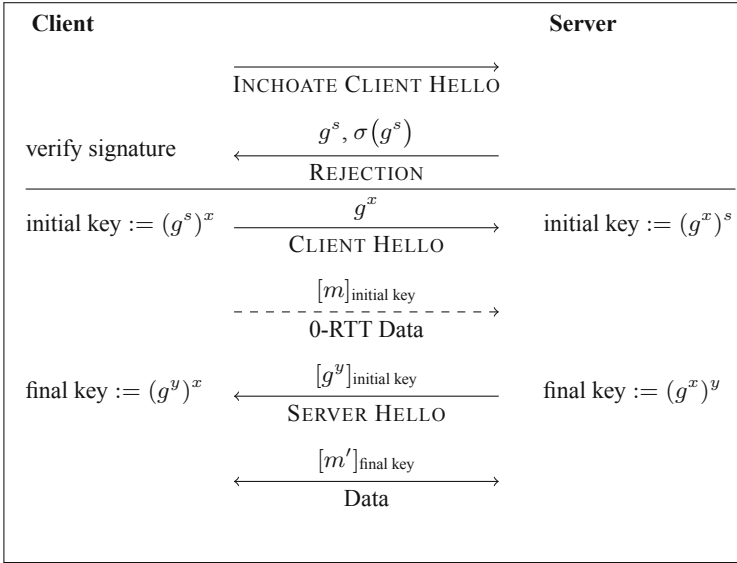


Fig. 1. Simplified QUIC handshake protocol. $\sigma(\cdot)$ denotes a signature, computed with the server’s long-lived signing key. If the server’s g^s is known, only the part below the horizontal divider is executed.

Because of the semi-static nature of the SERVER CONFIGURATION, the initial key derived from it is not forward-secure. To address this issue, a final key g^{xy} is derived from a DH share g^y with a freshly generated server exponent y . The server’s new DH share is embedded in a SERVER HELLO directed to the client. The client can derive the final key from the server’s new DH share and use it for all further communication. Note that the final key does provide forward security.

2.2 Bloom Filter Key Encapsulation Mechanisms

In this section we give a brief intuition on the main building block of our protocol. The implemented protocol is based on a puncturable key encapsulation mechanism (PKEM). A PKEM is closely related to a standard key encapsulation mechanism. In addition to the standard KeyGen, Encap and Decap algorithms for key generation, encapsulation and decapsulation, respectively, there is a Punc algorithm for puncturing in a PKEM. Given a secret key sk and a ciphertext C this algorithm outputs a modified secret key sk' . This modified secret key sk' has the property that it cannot be used to decapsulate C again. Therefore, by repeatedly calling the Punc algorithm, the set of ciphertexts which cannot be decapsulated, can be extended successively. Due to the practical inefficiency of known PKEM constructions, we base our protocol on a special variant called *Bloom filter key encapsulation mechanism* (BFKEM) [15]. A BFKEM introduces a correctness error in order to achieve highly efficient puncturing when compared

to other known PKEM constructions. However, this error can be made arbitrarily small.

Definition 1 (BFKEM). *A Bloom filter key encapsulation scheme BFKEM with key space \mathcal{K} is a tuple $\text{BFKEM} = (\text{KeyGen}, \text{Encap}, \text{Punc}, \text{Decap})$ of PPT algorithms:*

$\text{BFKEM.KeyGen}(1^\lambda, n, p)$. *On input of a security parameter λ , a number of expected punctures n and a bound p on the failure probability of decapsulate outputs a secret key and a public key (sk, pk) (where \mathcal{K} is implicitly defined by pk).*

$\text{BFKEM.Encap}(\text{pk})$. *On input of a public key pk outputs a ciphertext C and a symmetric key K .*

$\text{BFKEM.Punc}(\text{sk}, C)$. *On input of a secret key sk and ciphertext C outputs a modified secret key sk' .*

$\text{BFKEM.Decap}(\text{sk}, C)$. *On input of a secret key sk and ciphertext C outputs a symmetric key K or \perp if decapsulation fails.*

For the formal correctness and security definitions, we refer the reader to [15].

Previous works [15, 20] showed that a puncturable KEM can be used to construct 0-RTT protocols with full forward security, by using the `Punc` algorithm. A client sends a KEM ciphertext to the server. The server receives the ciphertext, decrypts it, and then calls `Punc`. After puncturing, it is impossible to decapsulate that ciphertext again, even given the punctured secret key. Thus, even if the server is compromised at some point in time, a previous KEM ciphertext can not be decrypted by the adversary.

Parametrization of BFKEM. A BFKEM needs two parameters for instantiation. By choosing these parameters according to the application at hand, the secret key size as well as the failure probability of `Decap` can be controlled. The first parameter is the expected number of invocations of the `Punc` algorithm n over the lifetime of the public key. The second parameter is the desired bound p on the failure probability of decapsulate which holds while fewer than n punctures have been executed.

A secret key of a BFKEM typically consists of a large array of subkeys. The optimal size m for this array can be derived from the parameters n and p . More precisely, it is given by $m = -n \ln p / (\ln 2)^2$ [15]. Thus, apart from choosing the lifetime according to the application, instantiation of a BFKEM is essentially a trade-off between the failure probability and the secret key size.

2.3 The Implemented Handshake Protocol

The implemented handshake protocol is based on the generic 0-RTT protocol of Günther *et al.* [20]. In the following, we describe a simplified version of this protocol using the aforementioned BFKEM. A visualization of this protocol is shown in Fig. 2.

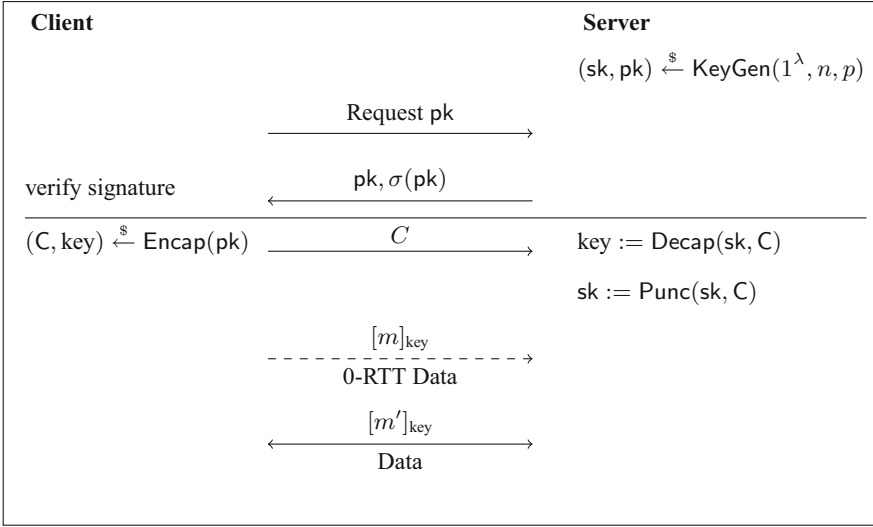


Fig. 2. Simplified version of the implemented handshake protocol. $\sigma(\cdot)$ denotes a signature, computed with the server’s long-lived signing key. If the server’s \mathbf{pk} is known, only the part below the horizontal divider is executed

Upon a server’s initialization, it uses the KeyGen algorithm to generate a BFKEM key pair $(\mathbf{sk}, \mathbf{pk})$. Since the client initially does not possess any information about the server’s key material, it needs to initiate a 1-RTT connection. When a client connects for the first time, the server transmits its BFKEM public key as well as a signature of the public key to the client. Once the client receives this message, it verifies the signature of the server’s public key and stores it for further processing if the signature is valid. The client can reuse the previously stored public key in subsequent connections to the same server. If the server’s public key has been replaced, the client needs to repeat the above steps.

The client proceeds with a 0-RTT connection. First, the client invokes the encapsulation algorithm to generate both a session key and a ciphertext. The session key is used to encrypt the 0-RTT data. Afterwards, the ciphertext and the encrypted 0-RTT data are sent to the server. Upon receiving the ciphertext, the server invokes the decapsulation algorithm to retrieve the session key. The server executes the puncturing algorithm to puncture its secret key before decrypting the encrypted 0-RTT data with the session key. Henceforth, the session key can be used for further communication.

2.4 Instantiation of the BFKEM

In [14] four different BFKEM constructions were presented. We have decided to use the one based on identity-based broadcast encryption (IBBE). In contrast to the other three, this one is able to achieve constant size ciphertexts while keeping public and secret keys reasonably small. As suggested in [14] we instantiated the

IBBE scheme with the construction presented by Delerablée [13]. The ciphertexts in her scheme are constant size and thus the BFKEM achieves constant size ciphertexts as well. Additionally, the delegated secret keys in the scheme by Delerablée consist only of a single group element. As the secret key in the construction of BFKEM based on IBBE consists of a large array of secret keys from the IBBE scheme, the secret key of the BFKEM benefits from the small secret keys from Delerablée’s scheme.

2.5 Failure Probability and Key Exhaustion of BFKEMs

In contrast to classical key encapsulation schemes, in a BFKEM the decapsulate algorithm has a probability of failure. A failure is intended for ciphertexts on which the secret key was already punctured, as this is the tool to achieve forward security. However, decapsulate may also fail on input of a ciphertext on which the secret key was not yet previously punctured.

In Fig. 3 we simulated a BFKEM for different values of the desired failure probability p while fixing the number of expected punctures n over the lifetime of the public key. For each simulation, we consecutively generate a fresh ciphertext, decapsulate this ciphertext, and then puncture the secret key on that same ciphertext. We do this until we are well above the threshold n . Each decapsulation failure is indicated by a vertical black line in the figure. For this simulation we fixed n to an exemplary value of 1500, however different values for n will result in a similar behaviour. It can be observed that after n punctures the bound p on the failure probability does not hold any more. Thus, exceeding the

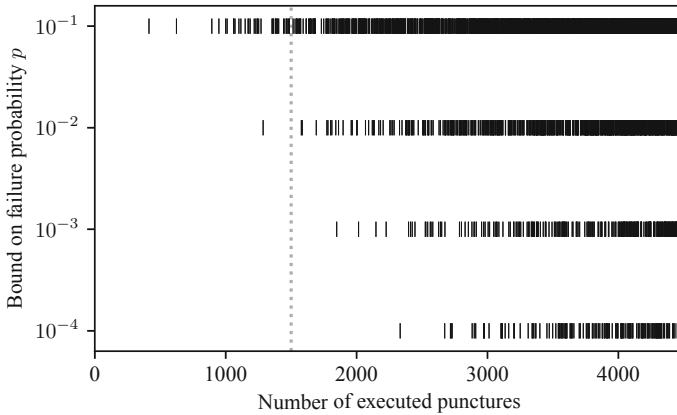


Fig. 3. Trend of decapsulation failures over number of executed punctures. Consecutive decapsulations of fresh ciphertexts and punctures of the secret key were simulated in a BFKEM for different values of the bound p on the failure probability valid until n punctures have been executed. Expected number of punctures over public key lifetime is fixed to $n = 1500$, which is indicated in the figure by the dotted line. A decapsulation failure is marked by a vertical black line.

expected number of punctures invalidates the guarantee on the failure probability of decapsulate. However, the bound p can be made arbitrarily small, and n can be made large by a suitable choice of Bloom filter parameters. We explain our choice of parameters below.

3 Security

A security model to formally analyze a 0-RTT key exchange was introduced by Günther *et al.* [20, Def. 11]. The authors additionally give a generic construction based on PKEM to build a 0-RTT key exchange with replay resilience and server-only authentication [20, Def. 12]. To account for the non-negligible correctness error of BFKEM the correctness property of that model was slightly adjusted in [15], however the authors argue that the generic construction can be instantiated with BFKEM without any changes.

The protocol we implemented as described in Sect. 2.3 resembles the generic 0-RTT protocol of Günther *et al.* [20] instantiated with a BFKEM as suggested in [15] and the maximum number of timesteps fixed to $\tau_{max} = 1$. The only difference is that the protocol from [20] assumes that a client knows the server's public key before starting a session. However, in practice this is not the case when a client connects to a server for the first time. For that reason, we assume the existence of a public key infrastructure which is used to transmit the server's public key to the client in an authenticated manner.

Günther *et al.* prove their generic 0-RTT protocol secure in their aforementioned security model under the assumption that the underlying BFKEM provides IND-CCA security [20, Thm. 2]. Derler *et al.* prove that their construction of BFKEM based on IBBE is IND-CCA-secure (resp. IND-CPA) if the underlying IBBE scheme is IND-sID-CCA-secure (resp. IND-sID-CPA) [14, Thm. 5]. Delerablée proves her construction of an IBBE scheme to be IND-sID-CPA-secure [13, Thm. 1], however, this is not sufficient for the protocol from [20]. Therefore, in order to achieve IND-CCA security for the BFKEM, Derler *et al.* [14] suggest to apply the Fujisaki-Okamoto transformation [16] to the BFKEM if the underlying IBBE scheme only provides IND-sID-CPA security. This transformation requires to encapsulate again within the decapsulate algorithm and thus adds significant computational overhead to decapsulation. In order to improve efficiency, we instead applied the transformation by Canetti, Halevi and Katz [10] (CHK transformation) to achieve IND-sID-CCA for the scheme by Delerablée as suggested in [13]. For a formal security proof of this modified CHK transformation, we refer the reader to [17].

During encapsulation, this requires the client to generate a fresh key pair of a sEUF-1-CMA-secure one-time signature scheme as well as to sign the ciphertext. The server then additionally has to verify this signature during decapsulation. Hence, the overhead added by this transformation depends on the used signature scheme. We instantiate it with the Boneh-Lynn-Shacham signature scheme [7] which is known to be EUF-CMA-secure and provides short signatures. Since there is exactly one valid signature for every public key and message pair this also guarantees sEUF-CMA (and thus sEUF-1-CMA) security.

4 Implementation

Our implementation is based on QUIC version 43. To be precise, we used the most up-to-date revision present on the Chromium Projects master branch [2] when we began implementing our changes, which was commit 2d376507075d on 31st of May 2018. We verified that our modifications are still applicable in the current version of QUIC as all changes since then have been of cosmetic nature only and did not change how the handshake is executed.

Goal of our Implementation. Our goal is to implement the protocol described in Sect. 2.3 into a real-world application. As our major design decision we agreed on implementing this protocol without removing the possibility to perform the QUIC handshake. As a consequence, benchmarking results are better to compare.

Modifying the QUIC Protocol. In the following, we traverse the message flow of the QUIC protocol while pointing out which parts we have modified.

- Initially, the server has to set up its keys for key exchange with the clients. In QUIC, the server generates the `SERVER CONFIGURATION` with the public DH share. In our implementation, the server instead uses the `KeyGen` algorithm to generate a BFKEM key pair (`sk`, `pk`). Note that the Bloom filter key material is (similar to QUIC’s `SERVER CONFIGURATION`) *medium-lived*².
- Upon a client’s first connection to a server, both parties agree on a common protocol version consisting of the handshake protocol and the transport protocol version. To negotiate on the newly implemented protocol, we added an entry to the supported handshake protocols in the QUIC implementation.
- If a server receives an `INCHOATE CLIENT HELLO`, it responds with a `REJECTION`. Instead of a DH share included in the `SERVER CONFIGURATION`, now the server’s BFKEM public key is embedded within this message. We removed the `SERVER CONFIGURATION`, since we do not use a DH key exchange for our implementation.
- Additionally, QUIC offers two algorithms to sign and verify a `SERVER CONFIGURATION`: `ECDSA-SHA256` and `RSA-PSS-SHA256`. We reused this functionality to sign the server’s public key instead of the `SERVER CONFIGURATION`. Note that the signing keys of the server are *long-lived*.
- Once a client receives a server’s public key it verifies its signature. Then a key as well as a ciphertext are computed by using the `Encap` algorithm. This key is used as a premaster secret, which is given to QUIC’s key derivation function. A freshly generated client nonce is included as salt. The derivation function uses HMAC with SHA-256 to generate two pairs of session keys and initialization vectors. Consequently, we do not need to manually set any session key or initialization vector. Analogous to the server, the client does not send an additional DH share in its `CLIENT HELLO`. Instead of the DH

² The server may choose any lifetime for the Bloom filter key material by parametrizing the Bloom filter accordingly. We provide a concrete parametrization for our analysis in Sect. 5.1.

- share, the ciphertext is included in the CLIENT HELLO. The client nonce is also added to the CLIENT HELLO such that both parties use the same salt.
- When a server receives a CLIENT HELLO, the Decap and Punc algorithms are executed using the received ciphertext. The key computed by the Decap algorithm is passed to QUIC’s key derivation as a premaster secret and the received client nonce is added as salt. In contrast to the QUIC protocol, the server does not send an additional DH share in the SERVER HELLO to establish a forward-secure key. This step can be omitted since the key exchanged by the proposed protocol already provides forward security.
 - By default, QUIC provides two authenticated encryption with associated data algorithms: Galois Counter Mode with AES128 and Poly1305 with ChaCha20. Since we did not alter the key derivation function, but only changed its input, both of these algorithms can be chosen.

Handshake Protocol Errors. There are cases in which the normal flow of the handshake can be interrupted. For instance, a client may initiate a 0-RTT connection using an out-of-date server public key or the Decap algorithm may fail, leaving the server unable to extract the received key. In any of the above events, the server responds by sending a REJECTION containing an appropriate rejection reason to the client. These failures replace corresponding errors occurring with QUIC’s DH key exchange. Any other errors or rejection reasons remain untouched. The client restarts the 0-RTT connection based on an up-to-date server public key.

Cryptographic Primitives. We implemented the IBBE scheme by Delerablée in the C programming language using the RELIC toolkit [3] for arithmetic operations in bilinear groups including pairings. Building upon that, we implemented the BFKEM described in Section 2.4 in C++. We optimized both implementations for speed, using multi-threading and precomputation tables where applicable. Additionally, our implementation offers optional point compression for the IBBE secret keys where we only store one coordinate of an elliptic curve point. This cuts memory requirements for each IBBE secret key in half while slowing down the decapsulation algorithm as the discarded coordinate must be recomputed. To reduce the size of transmitted data, we apply point compression to both public key and ciphertext as well.

5 Analysis

In this section, we analyze the efficiency of our implementation. To do so, we first describe a measurement setup as well as metrics and methodologies used to conduct performance tests. Building upon that, we evaluate the efficiency of our implementation by comparing its performance to QUIC.

5.1 Measurement Setup

Scenario. We consider the following scenario in our analysis: Several clients connect to a single web server several times. All of them are running the modified QUIC implementation using the BFKEM described in Sect. 4. For the key generation on server side we need to parametrize the BFKEM. We parametrized the BFKEM such that over a public key lifetime of two days one request per second can be served while guaranteeing a bound on the failure probability for decapsulate of $p = 0.001$. The two days were chosen as this is the lifetime currently used for the SERVER CONFIGURATION of Google’s QUIC servers. Additionally, we disabled point compression for the secret key³.

Testbed. We execute all performance tests on a networked client-server environment consisting of three desktop machines, connected via Gigabit Ethernet on a single switch, without additional latency emulation between machines⁴. One machine is used as a server, the other ones as clients. For a large part of our measurements we perform stress testing on the server side, i.e. we need to be able to exhaust the computational resources of the server. To exhaust the server, several clients need to send their requests to the server simultaneously, whereby the exact number of required clients depends on the performance of both the server and the client machines.

In the resulting setup, we use an Intel Core 2 Duo E6600 @ 2.40 GHz with 4 GB RAM as a server and two Intel Core i5-6600 CPU @ 3.30 GHz with 16 GB RAM to host *several client instances in parallel*. Since we noticed high fluctuations in results when running a large number of distributed dedicated client machines simultaneously, we purposely decided to run the server on a *low-performance* machine, while the clients are running on *high-performance* machines. This makes it possible to reduce the number of clients that are required to exhaust the server, resulting in much less coordination needed between clients and less fluctuations in results. All machines run Debian 9.8 (stretch). We utilize the QUIC test server and test client applications included in the Chromium sources. Because their native capabilities did not meet our requirements for testing, we extended the test client by the following features:

1. The client is able to perform multiple sequential requests within one and the same execution of the performance test. This allows us to perform 1-RTT handshakes as well as 0-RTT handshakes. Further, we eliminated any additional overhead that comes with starting and terminating the client application over and over again.

³ Enabling point compression leads to a decrease of 19% in memory consumption on server side while increasing the computational load per decapsulation by roughly 6%.

⁴ All machines are located within the same room. Hence, the resulting network latency is significantly lower compared to real-world latencies between clients and servers, especially compared to the required computation time of the implemented protocol. Overall, the network latency does not influence our results and is thus neglected in the following sections.

2. The client is able to wait a given amount of time between sequential requests. The waiting time starts as soon as the previous request has completed.
3. The client is able to run for a given amount of time. Within this time span, requests are performed. When the timer expires, any ongoing request is finished and the client terminates. The number of requests that could be completed within the time span is counted at client termination.

5.2 Metrics and Methodology

For a performance comparison between our implementation and QUIC we conduct measurements on throughput, computational cost and memory consumption. We further analyze different handshake properties in more detail.

- **Throughput.** Throughput is measured in requests per second. We run two clients in parallel on each machine, i.e. in total four clients are used to generate the requests. All clients perform requests over a runtime of 30 s. After 30 s, we inspect how many requests could be completed within this time span and reduce the result to one second. In order to obtain server-sided throughput limits, we vary the client-generated load by altering the waiting time between requests. Thus we are able to achieve different levels of offered load without changing the number of clients. We alter the waiting time in the range of 0 ms (no wait between requests) to 1000 ms (one second wait after each completed request). Note that since we are using four clients in parallel, within the frame of one waiting time a total amount of four requests is sent to the server.
- **Computational cost.** Computational costs are quantified in two ways. Firstly, we re-enact our throughput experiment, but additionally measure the server CPU utilization. To obtain measurement data on CPU utilization, we use the Linux performance monitoring tool *pidstat* [18]. We attach *pidstat* to the server process as soon as the clients start making requests and stop monitoring once all clients have finished. Secondly, we measure the CPU instructions per request. To measure the instruction count we utilize the Linux performance analyzing tool *perf* [1]. More precisely, we use the *perf stat* subcommand, which gives detailed information about the process that has been executed under *perf*'s supervision. Instructions per request are calculated as an average over 1000 requests.
- **Memory consumption.** We measure server-sided memory consumption. Again, we utilize *pidstat* to sample current memory consumption at a rate of one hertz. We track memory from server startup until it reaches a steady state, i.e. the server's memory does not increase any further and the server is ready to receive requests. From all samples we pick the maximum memory consumption.

In addition to the aforementioned metrics, we furthermore analyze the handshake regarding duration and size.

- **Handshake time.** We measure the time that is needed to complete a handshake. Time measurement begins when the client starts building the CLIENT

HELLO and ends when the SERVER HELLO is fully processed. We use QUIC logging functions to obtain the corresponding timestamps. The QUIC logs provide timestamps in the precision of one microsecond. We measure both 1-RTT and 0-RTT handshake times.

- **Handshake size.** We measure the bytes per handshake. The size of the handshake is calculated as the sum of all handshake message sizes. Values for message sizes are obtained from debug logs of the client and server application. Again, we measure both 1-RTT and 0-RTT handshake sizes.

In order to minimize any additional transmission and computational costs, we only transmit a small file of 100 bytes size with each request. To mitigate any bias resulting from transient noise either on network or on operating system level, all experiment results are averaged over ten repetitions.

5.3 Performance Comparison with QUIC

We compare our implementation with the original QUIC implementation. A summary of this is available in Table 1.

Table 1. High-level comparison of Google’s QUIC implementation and our modified version utilizing the implemented BFKEM (failure probability $p = 0.001$, expected number of punctures $n = 60^2 \cdot 24 \cdot 2 = 172800$).

Handshake	QUIC	Our implementation in QUIC
Forward Security	After 2 days	Always
Replay Resilience	No	Yes
Bytes per Handshake	1-RTT: 4027 0-RTT: 1358	1-RTT: 4755 0-RTT: 1188
Server CPU instructions per Request	13.57M	104.82M
Server memory usage	17.16 MB	658.28 MB
Handshake duration	1-RTT: 74.22 ms 0-RTT: 4.32 ms	1-RTT: 116.71 ms 0-RTT: 36.59 ms

Throughput. A throughput comparison between our implementation and QUIC is shown in Fig. 4. In our test setup, we achieve a maximum throughput of 34 requests per second for our implementation and 261 requests per second for QUIC. When only a small amount of requests is sent by the clients, achievable throughput between our implementation and QUIC behaves similarly. In this phase, the server can process all requests directly without delay. Only one request has to be processed at any time.

The higher the client-generated load, the more requests have to be handled in parallel. This has two effects: First, the processing time for each request increases, thus the achieved throughput on the server side diverges from the offered load of the clients. Second, when the server is computationally exhausted, the achieved throughput reaches a stable state. At this point, any additional requests sent by the clients do not lead to an increase in throughput on the server side. Since our implementation is considerably more computationally heavy, the server reaches its exhaustion level about eight times earlier compared to QUIC, resulting in an eight times lower throughput limit.

Computational Cost. In Fig. 4, CPU utilization of our implementation and QUIC is shown in comparison. We plot the CPU utilization as a function of the server throughput in the interval of 0 to 35 requests per second. In general, the CPU utilization increases linearly with the server throughput.

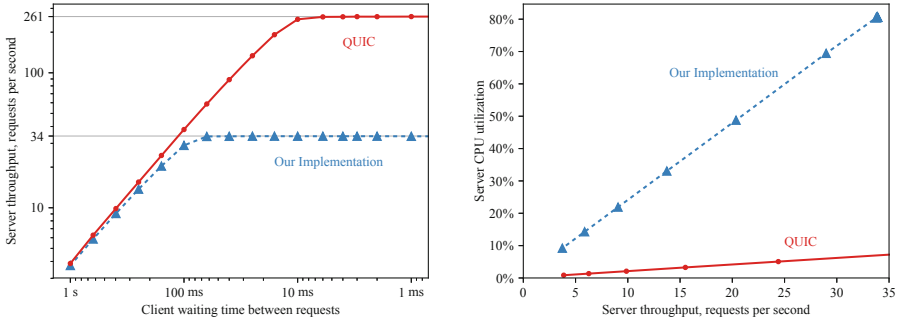


Fig. 4. The left figure shows the achievable server throughput for a given client-generated load. The measurement of our implementation is compared with the original QUIC implementation. The right figure shows the server CPU utilization for a given throughput. The measurement of our implementation is compared with the original QUIC implementation. Note that we plot in the range from 0 to 35 requests per second, which encloses the throughput limit of our implementation.

As expected from the increase in computations in our implementation, we experience a much higher CPU utilization for a given throughput in comparison to QUIC. For the lowest throughput of four requests per second we notice a CPU utilization of approximately ten percent for our implementation and less than one percent for QUIC. At the upper bound, we achieve a maximum of approximately 80% for our implementation and 50% for QUIC. Note that we run the server on a dual core machine in our experiments. Due to our multi-threading optimized implementation of the cryptographic primitives, we gain a higher CPU utilization at the server’s exhaustion limit. However, an ideal utilization of 100% is not achieved.

Having a look at the definite server CPU instructions executed per request, we can confirm a correlation between throughput limit and computational demands. As stated above, in our implementation we approximately achieve an eighth of the maximal throughput of QUIC. The same relation holds for the executed CPU instructions per request, i.e. for our implementation, approximately eight times more instructions have to be executed for each request. Measurements on executed instructions are given in Table 1.

Memory Consumption. In our test scenario, the server’s memory consumption reaches its maximum at 658 MB, resulting in almost 40 times higher memory requirements as compared to QUIC. In our implementation, server side memory

consumption is heavily influenced by the secret key size, which in turn depends on the choice of the BFKEM parameters.

Figure 5 shows the server’s memory consumption for a different number of expected punctures n over the lifetime of the public key and a fixed bound on the failure probability $p = 0.001$. The memory consumption then scales linearly with n as a larger secret key array size m is required to guarantee the chosen bound on the failure probability. Note that the measurements have been done with the implementation described in Sect. 4, i.e. we are using the BFKEM based on the IBBE scheme by Delerablée [13]. Therefore, the concrete measured values may differ when using another scheme than the one by Delerablée for instantiation. However, the scaling is independent from the scheme used for instantiation and is always linear.

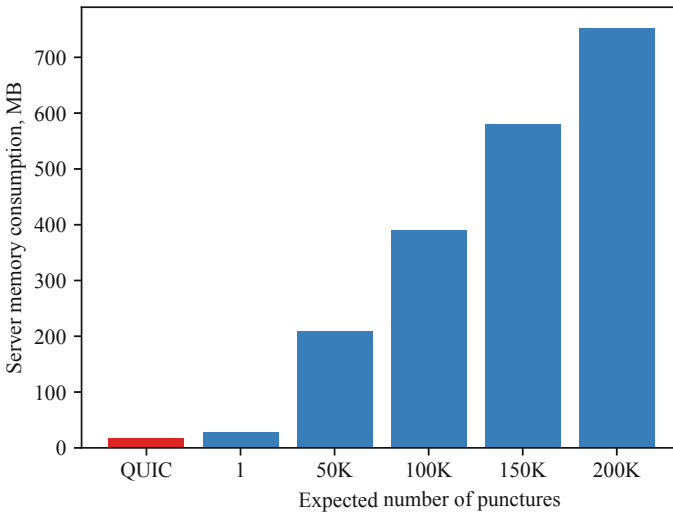


Fig. 5. Server memory consumption for different expected number n of punctures over the lifetime of the public key and fixed bound on the failure probability $p = 0.001$ with point compression disabled. The linear increase of memory consumption in n is due to the larger secret key array size m required to guarantee the chosen bound on the failure probability. The memory consumption of the QUIC server is shown for reference.

Handshake Analysis. We compare size and duration of 1-RTT and 0-RTT handshakes. In general, we notice an increase of 18% in size for 1-RTT handshakes and a decrease of 13% in size for 0-RTT handshakes when comparing our implementation to QUIC. Differences in handshake sizes are emerging from two handshake messages: the REJECTION and the SERVER HELLO. The REJECTION message increased by 898 bytes, primarily caused by replacing the SERVER CONFIGURATION with the server’s BFKEM public key. At the same time, the SERVER HELLO

in our implementation is reduced by 170 bytes due to removed information such as the DH share.

Measurements of the handshake duration correlate with the results of computational costs as described above. Due to the large increase in computational demands, both 1-RTT and 0-RTT handshakes require more time to be completed. Most notably in 0-RTT handshakes, added computations for encapsulation and decapsulation have remarkable impact on the resulting handshake duration. As a consequence, a 0-RTT handshake in our implementation takes approximately eight times longer as compared to QUIC. For a 1-RTT handshake, a large proportion of the overall duration is expended on signature verification. The duration of a 1-RTT handshake of our implementation therefore only differs by a factor of 1.6 in relation to QUIC.

Definite measurements on handshake size and duration are given in Table 1.

6 Conclusion

We have compared the 0-RTT key exchange implemented in QUIC with our implementation of a fully forward-secure 0-RTT key exchange. Despite the use of computationally heavy building blocks, such as pairings, the server CPU load increased only approximately 8 times, with a corresponding reduction in the achievable number of handshakes per second. The sizes of the handshake messages differ only by a few hundred bytes. These differences are not observable⁵ on the wire, which means that transmission times on the network are not affected by our changes. While the size of the secret key on the server side is significant, as it may grow to hundreds of megabytes and more, depending on the desired lifetime of the key and the acceptable failure probability of the key exchange, we think this is tolerable for modern server deployments with moderate resources.

The takeaways for protocol design depend on the design goal. Replay resilience and forward security may be considered worth the reduction in performance. Most importantly, the increased computation time of the 0-RTT handshake is still lower than that of a full 1-RTT handshake in the QUIC protocol. This means that even when the improvements by the reduced number of round trips, which will vary depending on network speed and latency, are not taken into account, the forward-secure 0-RTT mode is still preferable over 1-RTT from a latency perspective, as it reduces latency by roughly 50%.

Our analysis has shown that with currently available mechanisms, forward-secure 0-RTT handshake protocols can be considered practical. The performance of such a key exchange in real-world applications is worse than that of non-forward-secure 0-RTT protocols, but despite the measured increased computation times and computation load, it remains a viable alternative. Certainly, further improvements will be necessary if such protocols are supposed to gain widespread adoption.

⁵ Inspection in Wireshark revealed that messages are padded to occupy the full MTU size, canceling out small size differences.

Since puncturable encryption is a relatively novel area of research, we hope to see further constructions that might mitigate some drawbacks. Our work provides a benchmark that new constructions can be compared with. A possible approach may be to construct more efficient puncturable encryption schemes, which immediately yield a more efficient 0-RTT key exchange.

References






1. perf: Linux profiling with performance counters. https://perf.wiki.kernel.org/index.php/Main_Page
2. The Chromium Project. <https://www.chromium.org/>
3. Aranha, D.F., Gouvêa, C.P.L.: RELIC is an Efficient LIBrary for Cryptography. <https://github.com/relic-toolkit/relic>
4. Aviram, N., Gellert, K., Jager, T.: Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 117–150. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_5
5. Belshe, M., Peon, R., Thomson, M.: Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, IETF, May 2015. <http://tools.ietf.org/rfc/rfc7540.txt>
6. Belshe, M., Peon, R.: SPDY Protocol - Draft 3.1. Technical report, Google (2013), <https://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3-1>
7. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_30
8. Boyd, C., Gellert, K.: A modern view on forward security. *Comput. J.* (2020). <https://doi.org/10.1093/comjnl/bxaa104>
9. Brutlag, J.: Speed matters (2009). <https://ai.googleblog.com/2009/06/speed-matters.html>
10. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_13
11. Chang, W.T., Langley, A.: QUIC crypto (2014). https://docs.google.com/document/d/1g5nIXAikN_Y-7XJW5K45IbIHd_L2f5LTaDUDwvZ5L6g
12. Cheng, Y., Chu, J., Radhakrishnan, S., Jain, A.: TCP Fast Open. RFC 7413, IETF, December 2014, <http://tools.ietf.org/rfc/rfc7413.txt>
13. Delerablée, C.: Identity-based broadcast encryption with constant size ciphertexts and private keys. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 200–215. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76900-2_12
14. Derler, D., Gellert, K., Jager, T., Slamanig, D., Striecks, C.: Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. *Cryptology ePrint Archive, Report 2018/199* (2018) <https://eprint.iacr.org/2018/199>
15. Derler, D., Jager, T., Slamanig, D., Striecks, C.: Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 425–455. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_14
16. 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

17. Gellert, K.: Construction and security analysis of 0-RTT protocols. Ph.D. thesis, University of Wuppertal, Germany (2020). <https://doi.org/10.25926/eg6a-6059>
18. Godard, S.: Performance monitoring tools for Linux. <https://github.com/sysstat/sysstat>
19. Green, M.D., Miers, I.: Forward secure asynchronous messaging from puncturable encryption. In: 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015, pp. 305–320. IEEE Computer Society Press (2015)
20. Günther, F., Hale, B., Jager, T., Lauer, S.: 0-RTT key exchange with full forward secrecy. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10212, pp. 519–548. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56617-7_18
21. Iyengar, J., Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport. Draft draft-ietf-quic-transport-18, IETF, January 2019. <http://tools.ietf.org/id/draft-ietf-quic-transport-18.txt>
22. Lauer, S., Gellert, K., Merget, R., Handirk, T., Schwenk, J.: T0rtt: non-interactive immediate forward-secret single-pass circuit construction. Proceedings on Privacy Enhancing Technologies 2020(2), 336–357 (2020). <https://content.sciendo.com/view/journals/popets/2020/2/article-p336.xml>
23. Linden, G.: Marissa Mayer at Web 2.0 (2006). <https://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html>
24. Lychev, R., Jero, S., Boldyreva, A., Nita-Rotaru, C.: How secure and quick is QUIC? Provable security and performance analyses. In: 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015, pp. 214–231. IEEE Computer Society Press (2015)
25. MacCarthaigh, C.: Security Review of TLS 1.3 0-RTT. <https://github.com/tlswg/tls13-spec/issues/1001>. Accessed 29 Jul 2018
26. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (2018). <https://rfc-editor.org/rfc/rfc8446.txt>
27. Roskind, J.: Quick UDP internet connections: Multiplexed stream transport over UDP (2012). <https://docs.google.com/document/d/1RNHkx-VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit>
28. Strigeus, L., Hazel, G., Shalunov, S., Norberg, A., Cohen, B.: uTorrent transport protocol. Technical report, BEP29, BitTorrent.org (2009). http://www.bittorrent.org/beps/bep_0029.html
29. Thomson, M., Turner, S.: Using TLS to Secure QUIC. Internet-Draft draft-ietf-quic-tls-29, Internet Engineering Task Force, June 2020. <https://datatracker.ietf.org/doc/html/draft-ietf-quic-tls-29>, work in Progress

Elliptic Curves



Semi-commutative Masking: A Framework for Isogeny-Based Protocols, with an Application to Fully Secure Two-Round Isogeny-Based OT

Cyprien Delpech de Saint Guilhem^{1,2} , Emanuela Orsini¹ ,
Christophe Petit^{3,4} , and Nigel P. Smart^{1,2}  

¹ imec-COSIC, KU Leuven, Leuven, Belgium

{cyprien.delpechdesaintguilhem, emanuela.orsini, nigel.smart}@kuleuven.be

² Department Computer Science, University of Bristol, Bristol, UK

³ School of Computer Science, University of Birmingham, Birmingham, UK
christophe.f.petit@gmail.com

⁴ Département d'informatique, Université libre de Bruxelles, Brussels, Belgium

Abstract. We define semi-commutative invertible masking structures which aim to capture the methodology of exponentiation-only protocol design (such as discrete logarithm and isogeny-based cryptography). We give an instantiation based on the semi-commutative action of isogenies of supersingular elliptic curves, in the style of the SIDH key-exchange protocol. We then construct an oblivious transfer protocol using this new structure and prove that it UC-securely realises the oblivious transfer functionality in the random-oracle-hybrid model against passive adversaries with static corruptions. Moreover, we show that it satisfies the security properties required by the compiler of Döttling et al. (Eurocrypt 2020), achieving the first fully UC-secure two-round OT protocol based on supersingular isogenies.

1 Introduction

Since its beginnings, isogeny-based cryptography has progressed in several directions. First, that of protocol design, where primitives such as key-exchange and identification protocols [17, 20, 27] or signature schemes [24], have already been constructed. Secondly, in the understanding of the concrete security of the computational assumptions [23]. Finally, in the implementation methods for such protocols [3, 15, 19].

Whilst development of discrete-logarithm-based protocols has been rich, in terms of number of primitives, in the context of isogeny-based systems there has been less success. One reason is that the subtleties of isogeny-based primitives can be counter-intuitive (and even dangerous when misunderstood [22]). In particular, as noted in [17, 27], isogeny-based systems lack the commutative property which is often exploited in discrete-logarithm-based cryptography. Furthermore, the space of computational problems and their precise formulation is still shifting.

Supersingular isogeny-based protocols have attracted increasing attention mainly for their potential for post-quantum cryptography. In this direction some recent works [4, 7, 38] have proposed oblivious transfer (OT) protocols based on the hardness of supersingular isogeny problems. OT, originally introduced by Rabin in 1982 [34], is a fundamental primitive that has been proved complete for both two-party and multi-party computation, and has been used as building block in many efficient protocols [28, 31, 39]. Due to earlier interest in lattice-based and code-based cryptography, there have already been post-quantum OT protocols [5, 6, 32] based on the LWE, LPN and McEliece assumptions.

As well as underlying security assumptions, when we consider the state-of-the-art in post-quantum OT protocols we also need to take into account different factors, such as the security model and round complexity. Indeed, one of the most desirable properties, is having OT protocols with high security guarantees and only two rounds of communication. However, this is very hard to achieve and especially in the malicious setting, when one of the parties involved in the computation can arbitrarily deviate from the protocol. Indeed two-round OT with simulation based security is impossible in the plain model [26], and we need to rely on setup assumptions such as a common reference string or a random oracle.

Our Contribution. We consider a new approach for studying isogeny-based constructions by defining a new general framework for exponentiation-only protocols. We then apply this new structure and describe a simple oblivious transfer protocol with high security guarantees and minimal round complexity.

Semi-commutative Masking. We define new structures called *semi-commutative invertible masking schemes* to capture the exponentiation-only restriction of isogeny-based protocols and help draw out parallels with discrete-logarithm-based protocols. These also capture the absence of full commutativity in supersingular isogenies within a framework that is notationally simpler. In the full version, we show that these structures can also be realised in the discrete logarithm-based setting and in the setting of class group actions on endomorphism rings [12]. Moreover, we define generic computational problems for our structure and show that these correspond closely to the existing problems in the literature. The combination of our new structure together with instantiation-independent computational problems enables a clearer protocol design methodology. Furthermore, we believe that the hardness assumptions that we present can be extended to ones where more elements are given as a challenge (for example as used in pairing-based crypto). Such extended assumptions may enable the generic construction of schemes and protocols with richer functionalities as they have in the discrete-logarithm setting.

Isogeny-Based Oblivious Transfer. We illustrate the advantage of our framework describing a new two-round OT protocol constructed from our masking schemes. It achieves universal composability (UC) security against passive adversaries

with static corruptions in the random oracle model (ROM). In the full version we also show a second construction which is an adaptation of the key-exchange based protocol of Chou and Orlandi [14] to the “exponentiation-only” setting. Notably, our new structure allows us to provide a single proof of security for each protocol which is then valid for different instantiations of the masking scheme.

UC-Secure Isogeny-Based Two-Round OT. This only provides a two-round passively secure protocol, however we also show how to obtain a two-round maliciously secure protocol. The known methods for maliciously-secure OT are either based on zero-knowledge proofs or on “lossy” encryption schemes [32], which we don’t know how to instantiate using isogeny-based constructions and/or without increasing the round complexity. In [18], Döttling et al. introduced a general compiler to transform a rather weak and simple two-round *elementary*-OT (eOT), to a fully UC-secure two-round OT, providing also two instantiations: one based on the Computational Diffie-Hellman (CDH) problem and one on the Learning Parity with Noise (LPN) problem. We show (in Appendix 6) that our protocol satisfies the security requirements of this compiler, establishing the feasibility of two-round UC-secure OT based on semi-commutative masking, and more in particular on supersingular isogenies assumptions. In fact, we achieve the stronger notion of *search*-OT (sOT) security which means that Döttling et al.’s expensive transformation from eOT to sOT is not required for our protocol. To do so, we introduce a new problem for our masking scheme, called **ParallelDouble** (Definition 13), that is comparable to the one-more static CDH problem (where the adversary has access to both a challenger and a helper oracle and has to solve one more challenge than it was helped on).

Related Work. Since De Feo and Jao’s work [17, 27], others have explored different directions of supersingular isogenies [2, 3, 12, 15, 19–21, 23, 24, 30, 35]. However, to the best of our knowledge, our work is the first to present a framework for “exponentiation-based” protocols which unifies supersingular isogenies with previous constructions and also provides a separation between protocol design and analysis of computational assumptions. While we only present an OT protocol in this work, we believe that most of the works stated above can be formulated within our framework.

Recent works, concurrent and posterior to ours, have also proposed OT protocols based on supersingular isogenies [4, 8, 38]. The first describes an instantiation which is comparable to ours, especially regarding the computation of inverses and the question of the Weil pairing. It also proposes two protocols inspired by the same exponentiation-based approach and constructed from the same key-exchange and key-transport mechanisms. However, thanks to our new structure, our protocols better refine and separate the required computations. The OT protocol that we describe in this current paper fixes the two elements it requires for all instances, thus reducing the exchange to two flows – the best that can be hoped for, and the maximum allowed for Döttling et al.’s transformation to achieve UC security – instead of three, and it shifts the burden

Functionality \mathcal{F}_{OT}

PARAMETER: n length of the bit-strings

- Upon receiving $(P_S, \text{sid}, m_0, m_1)$ from P_S , check if a (sid, c) was previously stored. If yes, send m_c to P_R ; if not, store (sid, m_0, m_1) and continue to run.
- Upon receiving (P_R, sid, c) from P_R , check if a (sid, m_0, m_1) was previously stored. If yes, send m_c to P_R ; if not, store (sid, c) and continue to run.

Fig. 1. Oblivious transfer functionality

Functionality \mathcal{F}_{RO}

The functionality is parametrized by a domain \mathcal{D} and range \mathcal{R} . It keeps a list L of pairs of values, which is initially empty and proceeds as follows:

- Upon receiving a value $(\text{sid}, m), m \in \mathcal{D}$, if there is a pair $(m, \hat{h}), \hat{h} \in \mathcal{R}$, in the list L , set $h = \hat{h}$. Otherwise choose $h \xleftarrow{\$} \mathcal{R}$ and store the pair (m, h) in L .
- Reply to the activating machine with (sid, h) .

Fig. 2. Random oracle functionality

of computing the inverse to the Receiver. This reduces communication further and allows for only one inverse computation to be required. Using our masking structure, we also give another OT protocol, described in the full version, which separates the transmission of key material and choice material from the Sender to the Receiver. This permits the Sender to contribute to the final encryption key which is closer in spirit to the original key-exchange protocol. Vitse [38] also proposes an instantiation of her protocols from Kummer varieties; we leave it to further work to establish whether this could yield a new instantiation of our masking structure. Note, the works [4, 38] only prove security in the stand-alone and game-based models respectively, as opposed to our proofs in the UC model and there is no extension to malicious security.

Following the blueprint of previous works [5, 10], Branco et al. [8] achieve active security for OT at the cost of three additional rounds of communication. However, this requires the addition of a new mechanism which diverges from the “exponentiation-only” methodology. Furthermore, the security of their isogeny-based mechanism relies on assumptions that were only recently proposed [4] and have not yet been studied at length.

2 Preliminaries

We denote by λ the computational security parameter. We say that a function $f : \mathbb{N} \rightarrow \mathbb{N}$ is *negligible*, respectively *noticeable* (or non-negligible), if for every positive polynomial $p(\cdot)$ and all sufficiently large n it holds that $f(n) < \frac{1}{p(n)}$,

respectively $f(n) \geq \frac{1}{p(n)}$. We denote by $a \stackrel{s}{\leftarrow} A$ the uniform sampling of a from a set A , and computational and statistical indistinguishability by $\stackrel{c}{\approx}$ and $\stackrel{s}{\approx}$ respectively. We let $[n]$ denote the set $\{1, \dots, n\}$.

Symmetric Encryption. By $\mathcal{E} = \{(\text{KGen}_{\mathcal{E}}, \text{Enc}, \text{Dec}), (\mathcal{K}_{\mathcal{E}}, \mathcal{M}_{\mathcal{E}}, \mathcal{C}_{\mathcal{E}})\}$ we denote a symmetric encryption scheme, where $\mathcal{K}_{\mathcal{E}}, \mathcal{M}_{\mathcal{E}}, \mathcal{C}_{\mathcal{E}}$ are the key-space, message-space and ciphertext-space, respectively. We make use of the usual definition of IND-CPA security.

UC Security of Oblivious Transfer. We prove security of our protocols in the universal composition (UC) framework of Canetti [11], and assume familiarity with this. In particular, we prove in the full version that our protocol UC-realize the OT functionality \mathcal{F}_{OT} in the \mathcal{F}_{RO} -hybrid model, where \mathcal{F}_{OT} and \mathcal{F}_{RO} are presented in Figs. 1 and 2.

3 Semi-commutative Invertible Masking Structures

We first formally define our new masking structures and discuss some computational problems that arise in this setting. To help fix ideas we illustrate our masking structures with the case of discrete logarithms in a finite field \mathbb{F}_p , where $q = (p - 1)/2$ is prime and $g \in \mathbb{F}_p$ is an element of order q .

3.1 Masking Structure

A masking structure \mathcal{M} is defined over a set X . Each element $x \in X$ may have multiple *representations*, and we define R_x to be the set of representations of an element $x \in X$. (We require that it be efficient to recover x from any representation in R_x .) We denote the set of all such sets by $R_X = \{R_x\}_{x \in X}$. The sets of representatives are assumed to be disjoint, i.e. $\forall x, x' \in X$ s.t. $x \neq x'$, $R_x \cap R_{x'} = \emptyset$, and we define $R = \cup_{x \in X} R_x$ to be the set of all representatives. For example, if we take $X = \langle g \rangle \subset \mathbb{F}_p^*$, then the usual choice for R is to let $R_x = \{x\}$ for every $x \in X$; but one could also take a redundant representation with two elements letting $R_x = \{x, x + p\}$.

A *mask* is a function $\mu : R \rightarrow R$, and a masking set M is a set of such functions. In the discrete logarithm case, a natural candidate for M is a set indexed by elements in \mathbb{Z}_q^* which each give an explicit exponentiation algorithm on the set of representatives of the group elements X . A masking function $\mu \in M$ is said to be *invertible* if

$$\forall x \in X, \quad \forall r \in R_x, \quad \exists \mu^{-1} \in M \quad : \quad \mu^{-1}(\mu(r)) \in R_x. \tag{1}$$

Note, we only require that μ^{-1} outputs a representative in the same set R_x . If all elements $\mu \in M$ are invertible, then we say that the masking set M is *invertible*. In the discrete logarithm case, if μ corresponds to the map $g \mapsto g^a$, then μ^{-1} corresponds to the map $g \mapsto g^{1/a}$.

Data: $\mathcal{M} = \{X, R_X, [M_i]_{i=1}^n\}$, $\lambda \in \mathbb{N}$
Result: $\text{win} \in \{0, 1\}$
1 $r, \mu_0, \mu_1, i, \text{st} \leftarrow \mathcal{A}(1^\lambda)$ such that $r \in R$, $i \in [n]$, $\mu_0, \mu_1 \in M_j$, $j \neq i$;
2 $r_0 \leftarrow \mu_0(r)$, $r_1 \leftarrow \mu_1(r)$;
3 $b \xleftarrow{\$} \{0, 1\}$;
4 $\mu \xleftarrow{\$} M_i$;
5 $\tilde{r} \leftarrow \mu(r_b)$;
6 $\tilde{b} \leftarrow \mathcal{A}(1^\lambda, \text{st}, \tilde{r})$;
7 if $\tilde{b} = b$, then return $\text{win} = 1$ else return $\text{win} \xleftarrow{\$} \{0, 1\}$;

Fig. 3. The $\text{IND-Mask}_{\mathcal{A}, \mathcal{M}}$ security experiment

An *invertible masking structure* \mathcal{M} for a set X is then a collection of sets of representative R_X , along with a collection of invertible masking sets $[M_i]_{i=1}^n$, and we write $\mathcal{M} = \{X, R_X, [M_i]_{i=1}^n\}$. Such an invertible masking structure is said to be *semi-commutative* if

$$\forall i \neq j, \forall \mu \in M_i, \forall \mu' \in M_j, \forall r \in R, \mu(\mu'(r)) \in R_x \iff \mu'(\mu(r)) \in R_x. \quad (2)$$

In the discrete logarithm case, with M a set of exponentiation functions, $\mathcal{M} = \{X, R_X, [M, M]\}$ is straightforwardly semi-commutative.

3.2 Problems and Properties

We now present a distinguishing experiment and computational problems for masking structures. The precise security level of these depends from concrete instantiations and reductions to specific computational problems.

Definition 1 (IND-Mask security). We define the $\text{IND-Mask}_{\mathcal{A}, \mathcal{M}}$ experiment in Fig. 3 for a masking structure $\mathcal{M} = \{X, R_X, [M_i]_{i=1}^n\}$, and an arbitrary adversary \mathcal{A} . We say that \mathcal{M} is IND-Mask-secure if for all PPT adversaries \mathcal{A} , it holds that

$$\left| \Pr [\text{IND-Mask}_{\mathcal{A}, \mathcal{M}}(\lambda) = 1] - \frac{1}{2} \right| \leq \text{negl}(\lambda).$$

In the discrete logarithm setting, when $R_x = \{x\}$, the map $g \mapsto g^a$ for random $a \in \mathbb{Z}_q^*$ induces a random permutation of the group elements. Therefore for a secret a and two group elements g_0, g_1 , the distribution of g_b^a is perfectly uniform, independently of b . This shows that such an \mathcal{M} is perfectly IND-Mask-secure .

Note 1. In some settings (but not in the discrete logarithm one), it may be possible to distinguish the action of two masks that belong to separate masking sets. It is also possible that this difference is preserved under the action of a mask from a third masking set. Therefore, if an adversary was able to submit arbitrary r_0 and r_1 to the IND-Mask experiment, it could ensure that the difference between

them is preserved by the action of the randomly sampled μ and hence win the experiment with certainty. By forcing \mathcal{A} to submit a single $r \in R$ and two maps μ_0, μ_1 belonging to the same masking set M_j , the experiment prevents that strategy.

We also define to the following hard problems for semi-commutative invertible masking structures:

Definition 2. *Given a masking structure $\mathcal{M} = \{X, R_X, [M_i]_{i=1}^n\}$, we define the following computational problems:*

1. **Demask:** *Given (i, r, r_x) with the promise that $r_x = \mu_x(r)$ for a uniformly random $\mu_x \xleftarrow{\$} M_i$, return μ_x .*
2. **Parallel:** *Given (i, j, r, r_x, r_y) with the promise that $i \neq j$ and that $r_x = \mu_x(r)$ and $r_y = \mu_y(r)$ for uniformly random $\mu_x \xleftarrow{\$} M_i, \mu_y \xleftarrow{\$} M_j$, return $z \in X$ such that $\mu_x(r_y) \in R_z$.*
3. **ParallelInv:** *Given (i, j, r, r_x, r_y) with the promise that $i \neq j$ and that $r_x = \mu_x(r)$ and $r_y = \mu_y(r)$ for uniformly random $\mu_x \xleftarrow{\$} M_i, \mu_y \xleftarrow{\$} M_j$, return $z \in X$ such that $\mu_x^{-1}(r_y) \in R_z$.*
4. **ParallelEither:** *Given (i, j, r, r_x, r_y) with the promise that $i \neq j$ and that $r_x = \mu_x(r)$ and $r_y = \mu_y(r)$ for uniformly random $\mu_x \xleftarrow{\$} M_i, \mu_y \xleftarrow{\$} M_j$, return $z \in X$ such that either $\mu_x(r_y) \in R_z$ or $\mu_x^{-1}(r_y) \in R_z$.*
5. **ParallelBoth:** *Given $(i, j, r, r_{x_0}, r_{x_1}, r_y)$ with the promise that $i \neq j$ and that $r_{x_b} = \mu_b(r), \mathbf{b} \in \{0, 1\}$ and $r_y = \mu_y(r)$ for uniformly random $\mu_b \xleftarrow{\$} M_i, \mu_y \xleftarrow{\$} M_j$, return $z \in X$ such that either $\mu_{1-\mathbf{b}}^{-1}(\mu_{\mathbf{b}}(r_y)) \in R_z$ or $\mu_{\mathbf{b}}^{-1}(\mu_{1-\mathbf{b}}(r_y)) \in R_z$.*

To make explicit the given structure \mathcal{M} to which the (say) Demask problem refers, we write $\text{Demask}^{\mathcal{M}}$. The name “Parallel” is inspired by a similar problem defined by Couveignes [16].

We motivate these problems in the context of the discrete logarithm setting, where we take our masking structure as before to have $R_x = \{x\}$ and to have each M_i to be identical to the set of exponentiation maps indexed by \mathbb{Z}_q^* . We give a graphical intuition of these problems in Fig. 4.

- The Demask problem is, given (g, h) with the promise that $h = g^a$ for a random a , to return a . This is the discrete logarithm problem (DLP).
- Similarly, the Parallel problem is, given (g, g^a, g^b) for random a, b , to return $g^{a \cdot b}$ which is the computational Diffie-Hellman (CDH) problem.
- In the discrete logarithm setting, the ParallelInv problem is to compute $g^{b/a}$ given (g, g^a, g^b) . In the full version we show that this is equivalent to the Parallel problem. We note that this does not immediately hold in the abstract case, due to the absence of relation between r and $\mu^{-1}(\mu(r))$, but it can nonetheless be shown to hold for different instantiations.
- The ParallelEither problem is an instance where both the solutions to the Parallel and to the ParallelInv problems, for the same challenge, are accepted. Whilst it is immediate that the ParallelEither problem is at most as hard

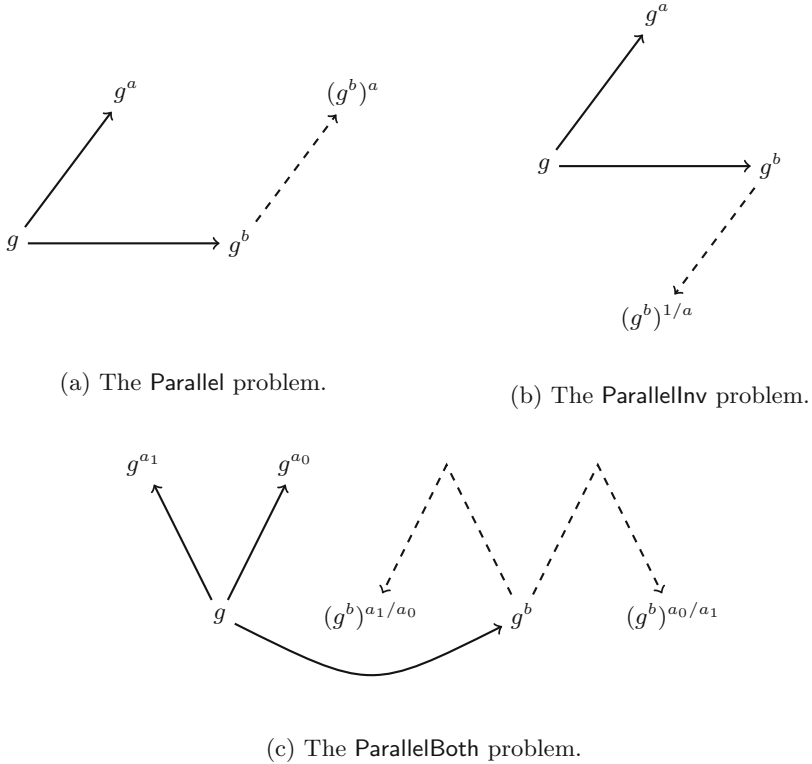


Fig. 4. Representations of computational problems.

as any of the other two, a formal reduction to show the reverse implication does not appear to be as trivial. We conjecture that in most settings, and in the discrete logarithm setting in particular, allowing for two possible answers which are both hard to compute on their own does not significantly decrease the hardness of the **ParallelEither** problem.

- The solution of the **ParallelBoth** problem can be seen as a combination of both **Parallel** and **ParallelInv** solutions together with the choice of the **ParallelEither** problem as is shown in Fig. 4c.

Indeed, one can first use a **Parallel** oracle to compute $\mu_b(r_y)$ for either $b \in \{0, 1\}$ and then use a **ParallelInv** oracle to compute $\mu_{1-b}^{-1}(\mu_b(r_y))$ which shows that **ParallelBoth** is at most as hard as those two problems. Similarly to the **ParallelEither** problem, we conjecture that in most settings the **ParallelBoth** will not be significantly easier as it requires solutions which are both hard to compute.

4 Instantiation from Supersingular Isogenies

To avoid a sub-exponential quantum attack vector [13], De Feo, Jao and Plût [17] consider the use of supersingular elliptic curves over the extension field \mathbb{F}_{p^2} whose full endomorphism ring is an order in a quaternion algebra and therefore non-commutative. In this section we summarize this approach succinctly, construct a semi-commutative masking structure from this setting and discuss the hardness of the induced problems.

4.1 Supersingular Isogenies over the Extension Field

Preliminaries. Let E_1 and E_2 be elliptic curves defined over a finite field \mathbb{F}_q . An *isogeny* $\phi : E_1 \rightarrow E_2$ over \mathbb{F}_q is a non-constant rational map over \mathbb{F}_q which is also a group homomorphism from $E_1(\mathbb{F}_q)$ to $E_2(\mathbb{F}_q)$. For the isogenies that we consider, we identify their degrees with the size of their kernels. Two curves E_1, E_2 are said to be *isogenous* over \mathbb{F}_q if there exists an isogeny $\phi : E_1 \rightarrow E_2$ over \mathbb{F}_q ; this holds if and only if $\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$. A set of elliptic curves over \mathbb{F}_q that are all isogenous to one another is called an *isogeny class*.

An *endomorphism* over \mathbb{F}_q of an elliptic curve E is a particular isogeny $E \rightarrow E$ over \mathbb{F}_{q^m} for some m . The set of endomorphisms of E together with the zero map, denoted $\text{End}(E)$, forms a ring under the addition, $\phi \oplus \varphi : P \mapsto \phi(P) + \varphi(P)$, and multiplication, $\phi \otimes \varphi : P \mapsto \phi(\varphi(P))$, operations. The full ring $\text{End}(E)$ is isomorphic to either an order in a quaternion algebra, in which case we say that E is supersingular, or to an order in an imaginary quadratic field, in which case we say that E is ordinary. Curves that are in the same isogeny class are either all supersingular or all ordinary. Here we focus on the supersingular case. All supersingular curves can be defined over the field \mathbb{F}_{p^2} for a prime p and for every prime $\ell \nmid p$ there exist $\ell + 1$ isogenies, up to isomorphism, of degree ℓ originating from any given supersingular curve.

Given a curve E and a subgroup K of E there is, up to isomorphism, a unique isogeny $\phi : E \rightarrow E'$ having kernel K and we therefore identify E' with the notation E/ϕ . Particularly, we will work with subgroups of the torsion group $E[m]$ for $m \in \mathbb{N}$ which is the group of points of E whose order divides m . When we also have that m^2 divides $\#E(\mathbb{F}_{p^2})$, we can always represent cyclic kernels by generators defined over \mathbb{F}_{p^2} .

Semi-commutativity. We introduce the notion of *semi-commutativity* present in this setting; the same notion is behind the SIDH key-exchange protocol [17] and we generalise it here. We discuss the case where \mathbb{F}_q is fixed to be \mathbb{F}_{p^2} where p is a prime of the form $\ell_1^{e_1} \ell_2^{e_2} \dots \ell_n^{e_n} \cdot f \pm 1$ for n small primes ℓ_1, \dots, ℓ_n and a small cofactor f . By construction, in each isomorphism class there is a curve E/\mathbb{F}_{p^2} such that the torsion group $E[\ell_i^{e_i}]$ contains $\ell_i^{e_i-1}(\ell_i + 1)$ cyclic subgroups of order $\ell_i^{e_i}$ (which each define a different isogeny).

To compute and publish a curve resulting from a secret isogeny, a party generates a secret key by selecting a random point K_i of order $\ell_i^{e_i}$ on a curve E

and computes a public curve by computing the unique isogeny with kernel $\langle K_i \rangle$ and publishing the domain curve $E/\langle K_i \rangle$. The issue here is that the structure of $\text{End}(E)$ no longer allows for arbitrary isogenies to commute and an analogue of the $(g^a)^b = (g^b)^a$ equality is not immediate. However, with isogenies of co-prime degrees some commutative structure remains.

To solve this, in addition to the curve E , the parties agree on bases $\{P_i, Q_i\}$ for each of the torsion groups $E[\ell_i^{e_i}]$. The semi-commutative structure then emerges since applying an isogeny of degree $\ell_i^{e_i}$ preserves the torsion groups $E[\ell_j^{e_j}]$ for $j \neq i$. Therefore, alongside publishing $E/\langle K_i \rangle$ for their secret isogeny ϕ_i , parties also publish $\{\{\phi_i(P_j), \phi_i(Q_j)\}_{j \neq i}\}$, the images under ϕ_i of the bases for the other torsion groups. By expressing their secret kernel as $K_j = [\alpha_j]P_j + [\beta_j]Q_j$ and applying α_j, β_j to $\{\phi_i(P_j), \phi_i(Q_j)\}$, the other party can then compute an isogeny $\varphi_j : E/\langle K_i \rangle \rightarrow E/\langle K_i, K_j \rangle$ which is “parallel” to the isogeny $\phi_j : E \rightarrow E/\langle K_j \rangle$ in the sense of Fig. 4a.

Whilst the two resulting curves $E/\langle K_i, K_j \rangle$ and $E/\langle K_j, K_i \rangle$ may not be identical, they will be isomorphic, and the parties can then take the j -invariants of their respective curves as an identical shared value.

The Weil Pairing. We recall here the notion of the *Weil pairing*. For any integer $m \in \mathbb{N}$, we let $\zeta_m = \{u \mid u^m = 1\} \subset \mathbb{F}_{p^2}^*$. For any curve E/\mathbb{F}_{p^2} , the Weil pairing is a map $e_m : E[m] \times E[m] \rightarrow \zeta_m$, that satisfies $e_m(\phi(P), \phi(Q)) = e_m(P, Q)^{\text{deg}(\phi)}$, where $\phi : E \rightarrow E'$ is any isogeny.

4.2 Masking Structure

To define a semi-commutative masking structure, we fix $p = \ell_1^{e_1} \ell_2^{e_2} \dots \ell_n^{e_n} \cdot f \pm 1$ as above. In this setting, there are five supersingular isogeny classes and we let X denote one of the two classes with curves E/\mathbb{F}_{p^2} with trace $t = p^2 + 1 - \#E(\mathbb{F}_{p^2}) \in \{-2p, 2p\}$; these two classes are the largest of the five [1].

Representatives. For each j -invariant $x \in X$, there is a canonical choice of curve E_x [36]. For each E_x we take the appropriate twist of the curve such that they belong to the same isogeny class. We define the set R_x of representatives as the set of tuples $(E_x, \{\{P_i, Q_i\}_{i \in [n]}\})$ where $\{P_i, Q_i\}$ is a basis of the torsion group $E_x[\ell_i^{e_i}]$ as above.

For a given curve and torsion order, there exists a deterministic and efficient algorithm $\text{Basis}(E, i)$ which outputs a basis $\{P_i, Q_i\} \subset E_x[\ell_i^{e_i}]$ [3, Section 3.2]; for each torsion order, we fix a generator $q_i \in \zeta_{\ell_i^{e_i}}$ such that for any curve E , $e_m(P_i, Q_i) = q_i$ for $\{P_i, Q_i\} \leftarrow \text{Basis}(E, i)$. This will be used to derive new torsion points when required, but these are still free to be modified under the action of isogenies. Hence for each x , there will be a unique choice of E_x but many choices of bases of torsion groups that originate from the deterministic one.

Masking Sets. We first observe that for any $K_i = [\alpha_i]P_i + [\beta_i]Q_i$ on E , the point $[m]K_i$, for $m \in (\mathbb{Z}/\ell_i^{e_i}\mathbb{Z})^*$, generates the same subgroup of $E[\ell_i^{e_i}]$. By defining the equivalence relation \sim_R by

$$(\alpha, \beta) \sim_R (\alpha', \beta') \iff \exists m \in (\mathbb{Z}/\ell_i^{e_i}\mathbb{Z})^* \text{ s.t. } (\alpha', \beta') = (m\alpha, m\beta),$$

we can then identify any such K_i with the equivalence class of (α_i, β_i) which we denote $[\alpha_i : \beta_i]$. We recall that the projective line $\mathbb{P}^1(\mathbb{Z}/\ell_i^{e_i}\mathbb{Z})$ is the set of equivalence classes $[\alpha_i : \beta_i]$ such that $\gcd(\alpha_i, \beta_i) = 1$.

Since K_i has exact order $\ell_i^{e_i}$, at least one of α_i and β_i must not be divisible by ℓ_i and hence the ideal of the ring $\mathbb{Z}/\ell_i^{e_i}\mathbb{Z}$ generated by α_i, β_i is always the unit ideal, i.e. the whole of $\mathbb{Z}/\ell_i^{e_i}\mathbb{Z}$. This implies that all the possible choices for K_i can be exactly identified with the points on the projective line $\mathbb{P}^1(\mathbb{Z}/\ell_i^{e_i}\mathbb{Z})$. We therefore define n masking sets $[M_i]_{i \in [n]}$ where each M_i is the projective line $\mathbb{P}_i := \mathbb{P}^1(\mathbb{Z}/\ell_i^{e_i}\mathbb{Z})$.

Masking Action. Computing the result of a mask $\mu(r) \in R_y$ on a representative $r \in R_x$ then consists in computing one of its representatives K_i in $E_x[\ell_i^{e_i}]$ and the isogeny $\phi_i : E_x \rightarrow E_x/\langle K_i \rangle$. Note that the curve $E_x/\langle K_i \rangle$ with j -invariant $y \in X$ may not be the same curve as the canonical choice E_y . However they will be isomorphic over \mathbb{F}_{p^2} , due to the appropriate choice of twist in the definition of our set R_y , and the isomorphism $\chi : E_x/\langle K_i \rangle \rightarrow E_y$ will be easy to compute.

To be able to compose isogenies in a semi-commutative way, computing $\mu(r)$ also requires computing the images of $\{\{P_j, Q_j\}\}$ for $j \neq i$ first under ϕ_i and then under the isomorphism χ to obtain bases of the torsion groups of E_y . It also requires generating a new basis for $E_y[\ell_i^{e_i}]$ using the $\text{Basis}(E_y, i)$ algorithm.

The output of the computation of the mask $\mu(r)$ is therefore the curve $E_y \xrightarrow{\chi} E_x/\langle K_i \rangle$ together with the basis points $\{\{\chi \circ \phi_i(P_j), \chi \circ \phi_i(Q_j)\}\}$ for $j \neq i$ and the output of $\text{Basis}(E_y, i)$.

Inverting the Mask. Since our masking sets M_i do not derive from a group structure, we do not have an immediate instantiation of an inverse operation. However, for every isogeny $\phi : E \rightarrow E'$ of degree ℓ , there is a unique dual isogeny $\hat{\phi} : E' \rightarrow E$ also of degree ℓ such that the composition is the multiplication-by- ℓ map: $\hat{\phi} \circ \phi = [\ell] : E \rightarrow E$. Whilst not a perfect inverse operation, in this setting the multiplication-by- $\ell_i^{e_i}$ map preserves the structure of the $\ell_j^{e_j}$ -torsion groups for all $j \neq i$ and that is all we require for semi-commutativity to hold.

Hence, given a kernel generator $K_i \in E[\ell_i^{e_i}]$ for some curve E , one can compute a generator of the image $\phi_i(E[\ell_i^{e_i}]) \subset E'[\ell_i^{e_i}]$ of the $\ell_i^{e_i}$ -torsion group under the isogeny ϕ_i defined by K_i and an appropriate isomorphism, to obtain $\hat{K}_i \in E'/\langle K_i \rangle$ which is a generator of the kernel of the unique dual isogeny $\hat{\phi}_i$.

Given a mask $\mu \in M_i = \mathbb{P}_i$ and elements r and $r' = \mu(r)$ with $r' \in (E', \{\{P_j, Q_j\}_{j \in [n]}\})$, computing the inverse μ^{-1} amounts to computing a point \hat{K}_i as above and expressing it as $(\hat{\alpha}_i, \hat{\beta}_i)$ in the deterministically generated basis for $E'[\ell_i^{e_i}]$ which can be done efficiently as is shown in [3]. This then allows us to

define μ^{-1} uniquely as $[\hat{\alpha}_i : \hat{\beta}_i] \in \mathbb{P}_i$, given μ and r . We note that the dependency of μ^{-1} on μ and r is consistent with the definition of the inverse of a mask as stated in Sect. 3.

Masking Structure. We formally define a masking structure in this setting.

Definition 3 (Masking structure from supersingular isogenies). *Let p be a prime defining the finite field \mathbb{F}_{p^2} as above, we define the masking structure $\mathcal{M}_p = \{X, R_X, [M_i]_{i \in [n]}\}$ where the individual components are defined as above.*

Lemma 1. *The masking structure \mathcal{M}_p of Definition 3 is semi-commutative.*

Proof. First we see that the elements of \mathcal{M}_p together with the action of any $\mu \in M_i$ on any r are well-defined. Then, since the composition of any isogeny with its dual results in an endomorphism of the starting curve, our method of inverting a given mask yields the same j -invariant regardless of the starting r or masking index i . Also, the semi-commutative property of our structure follows from the semi-commutative property of isogenies of co-prime degrees. Finally, the required efficiency of the computations for \mathcal{M}_p follows from the comments above regarding the computation of isogenies of smooth degrees and expression of points in arbitrary torsion bases. Equality in X and M_i and membership in X are immediate to check. □

4.3 Computational Problems

The problem landscape of the SIDH setting is still currently undergoing intense study from the community. Urbanik and Jao [37] have proposed a detailed presentation and study of the analogues of the discrete logarithm and CDH problems that arise from the SIDH key-exchange of De Feo, Jao and Plût [17]. Galbraith and Vercauteren also have written a survey of these problems [25], with a stronger focus on the mathematics of isogenies of elliptic curves.

Here we frame Urbanik and Jao’s discussion of these problems in [37, Section 4] in our setting that uses n distinct small primes ℓ_i . Whilst we give a very general presentation, in practice the OT scheme presented in this paper will only require $n = 2$, as in the case of the SIDH key-exchange. Our second OT protocol (described in the full version) will require $n = 3$, which constitutes only a small extension of the original setting.

The Isogeny Problem. In its simplest form, the intuition behind the security of isogeny-based cryptography is that it is hard to compute a hidden isogeny, up to isomorphism, when given only the initial and final j -invariants. The *general isogeny problem* can be stated as follows.

Definition 4 (General isogeny problem [25, Definition 1]). *Given j -invariants $j, j' \in \mathbb{F}_{p^2}$, return an isogeny $\phi : E \rightarrow E'$ (if it exists), where $j(E) = j$ and $j(E') = j'$.*

Given that the elements of X in the masking structure \mathcal{M}_p are the supersingular j -invariants of \mathbb{F}_{p^2} and that the elements of the masking sets M_i can be uniquely identified with isogenies between isomorphism classes, it would first seem that the Demask problem for \mathcal{M}_p can be instantiated as the general isogeny problem of Definition 4. To recover some commutative structure, however, we have to reveal the images of the bases of the torsion points. This constitutes significantly more information and therefore is conjectured to be an easier problem to solve [24,25,29,33].

Additional Information. This has led to the definition in the literature of a specific SIDH problem. Here we merge the definitions of [25] and [37] for the case of $n = 2$ small primes in the composition of p .

Definition 5 (2- i -isogeny problem [25, Def. 2][37, Prob. 4.1]). Let $i \in \{1, 2\}$ and let (E, P_1, Q_1, P_2, Q_2) be such that E/\mathbb{F}_{p^2} is a supersingular curve and P_j, Q_j is a basis for $E[\ell_j^{e_j}]$ for $j \in \{1, 2\}$. Let E' be such that there is an isogeny $\phi : E \rightarrow E'$ of degree $\ell_i^{e_i}$. Let P'_j, Q'_j be the images under ϕ of P_j, Q_j for $j \neq i$. The 2- i -isogeny problem, is, given $(E, P_1, Q_1, P_2, Q_2, E', P'_j, Q'_j)$, to determine an isogeny $\tilde{\phi} : E \rightarrow E'$ of degree $\ell_i^{e_i}$ such that $P'_j = \tilde{\phi}(P_j)$ and $Q'_j = \tilde{\phi}(Q_j)$.

This definition leads to the following natural generalisation which we show corresponds exactly to the computational problem that we need.

Definition 6 (n - i -isogeny problem). Let n be an integer, $i \in \{1, \dots, n\}$ and let $(E, \{P_j, Q_j\}_{j=1}^n)$ be a tuple such that E/\mathbb{F}_{p^2} is a supersingular curve and P_j, Q_j is a basis for $E[\ell_j^{e_j}]$ for $j \in [n]$. Let E' be such that there is an isogeny $\phi : E \rightarrow E'$ of degree $\ell_i^{e_i}$. Let $\{P'_j, Q'_j\}$ be the images under ϕ of $\{P_j, Q_j\}$ for $j \neq i$. The n - i -isogeny problem, for $i \in [n]$, is, given $(E, \{P_j, Q_j\}_{j=1}^n, E', \{P'_j, Q'_j\}_{j \neq i})$, to determine an isogeny $\tilde{\phi} : E \rightarrow E'$ of degree $\ell_i^{e_i}$ such that $P'_j = \tilde{\phi}(P_j)$ and $Q'_j = \tilde{\phi}(Q_j)$ for all $j \neq i$.

Lemma 2. Let $p = \ell_1^{e_1} \ell_2^{e_2} \dots \ell_n^{e_n} \cdot f \pm 1$ be a prime and let \mathcal{M}_p be a masking structure as defined in Definition 3. Then the Demask problem for \mathcal{M}_p is an instance of the n - i -isogeny problem.

Proof. The specification of i in (i, r, r_x) together with the random mask μ_x satisfies the promise of existence of an isogeny ϕ of degree $\ell_i^{e_i}$. Also, By definition of R_x for each $x \in X$ for \mathcal{M}_p , the representative r_x contains exactly the information of the curve E' together with the images of the appropriate torsion points. We note that r_x does not contain additional information as the basis points of $E'[\ell_i^{e_i}]$ are derived deterministically from E' . \square

Computational SIDH. The isogeny problems defined above can be viewed as the analogues of the discrete logarithm problem of computing an unknown exponent in the general case and in the specific SIDH setting. This naturally leads to an analogue of the CDH problem which is defined as follows in the case of $n = 2$.

Definition 7 (2-computational SIDH problem [37, Problem 4.3]). Let E, E_A, E_B be supersingular curves such that there exist isogenies $\phi_A : E \rightarrow E_A$ and $\phi_B : E \rightarrow E_B$ with kernels K_A and K_B and degrees $\ell_1^{e_1}$ and $\ell_2^{e_2}$ respectively. Let P_1, Q_1 and P_2, Q_2 be bases of $E[\ell_1^{e_1}]$ and $E[\ell_2^{e_2}]$ respectively, and let $P'_1 = \phi_B(P_1)$, $Q'_1 = \phi_B(Q_1)$ and $P'_2 = \phi_A(P_2)$, $Q'_2 = \phi_A(Q_2)$ be the images of the bases under the isogeny of coprime degree. The 2-computational SIDH problem is, given $(E, P_1, Q_1, P_2, Q_2, E_A, P'_2, Q'_2, E_B, P'_1, Q'_1)$, to identify the isomorphism class of the curve $E/\langle K_A, K_B \rangle$.

This problem can also be generalised in a natural way to the following which then yields the appropriate instantiation for our structure.

Definition 8 (n - i, j -computational SIDH problem). Let E, E_A, E_B be supersingular curves such that there exist isogenies $\phi_A : E \rightarrow E_A$ and $\phi_B : E \rightarrow E_B$ with kernels K_A and K_B and degrees $\ell_i^{e_i}$ and $\ell_j^{e_j}$ respectively with $i \neq j$. Let $\{P_k, Q_k\}$ be bases of $E[\ell_k^{e_k}]$, for $k \in [n]$, and let $P_k^A = \phi_A(P_k)$, $Q_k^A = \phi_A(Q_k)$, for $k \neq i$, and $P_k^B = \phi_B(P_k)$, $Q_k^B = \phi_B(Q_k)$, for $k \neq j$ be the images of the bases under the isogeny of coprime degree. The n - i, j -computational SIDH problem, for $i, j \in [n]$, is, given $(E, \{P_k, Q_k\}_{k \in [n]}, E_A, \{P_k^A, Q_k^A\}_{k \neq i}, E_B, \{P_k^B, Q_k^B\}_{k \neq j})$, to identify the isomorphism class of the curve $E/\langle K_A, K_B \rangle$.

Lemma 3. Let $p = \ell_1^{e_1} \ell_2^{e_2} \dots \ell_n^{e_n} \cdot f \pm 1$ be a prime and let \mathcal{M}_p be a masking structure as defined in Definition 3. Then the Parallel problem for \mathcal{M}_p is an instance of the n - i, j -CSIDH problem.

Proof. As for Lemma 2, the specification (i, j, r, r_x, r_y) of the Parallel problem for \mathcal{M}_p satisfies the promise of existence of the two isogenies of coprime degrees and contains all the required information on the images of the torsion bases. Also, the goals of the problems agree since the solution to the Parallel problem for \mathcal{M}_p requires $z \in X$ which is exactly the j -invariant which identifies the isomorphism class uniquely. Again, r_x and r_y do not contain additional information since the bases for the i th and j th torsion groups are computed deterministically. \square

Regarding the ParallelInv problem for \mathcal{M}_p , we do not have an immediate reduction to the Parallel problem. We discuss this in comparison to the instantiation from hard homogeneous spaces and also an interesting subtlety in the definitions of the CDH problem in the full version of this work. We nonetheless conjecture that, as they are very similar, the hardness of the ParallelInv problem is close to that of the Parallel problem. We similarly conjecture that the hardness of the ParallelEither and ParallelBoth problems is comparable to that of the Parallel and ParallelInv problems as no additional information is revealed and only similarly hard-to-compute solutions are required.

Decisional SIDH. Galbraith and Vercauteren also formalise a decisional variant of the SIDH problem in the case of $n = 2$.

Definition 9 (2-*i*-decisional SIDH problem [25, Definition 3]).

Let (E, P_1, Q_1, P_2, Q_2) be such that E/\mathbb{F}_{p^2} is a supersingular curve and P_j, Q_j is a basis for $E[\ell_j^{e_j}]$ for $j \in \{1, 2\}$. Let E' be an elliptic curve and let $P'_j, Q'_j \in E'[\ell_j^{e_j}]$ for $j \neq i$. Let $0 < d < e_i$. The 2-*i*-decisional SIDH problem is, given $(E, P_1, Q_1, P_2, Q_2, E', P'_j, Q'_j, d)$ for $j \neq i$, to determine if there exists an isogeny $\phi : E \rightarrow E'$ of degree ℓ_i^d such that $\phi(P_j) = P'_j$ and $\phi(Q_j) = Q'_j$.

As for the computational problems, we can generalise the above problem to our setting.

Definition 10 (*n*-*i*-decisional SIDH problem). Let $(E, \{P_j, Q_j\}_{j \in [n]})$ be such that E/\mathbb{F}_{p^2} is a supersingular curve and P_j, Q_j is a basis for $E[\ell_j^{e_j}]$ for $j \in [n]$. Let E' be an elliptic curve and let $P'_j, Q'_j \in E'[\ell_j^{e_j}]$ for $j \neq i$. Let $0 < d < e_i$. The *n*-*i*-decisional SIDH problem is, given $(E, \{P_j, Q_j\}_{j \in [n]}, E', \{P'_j, Q'_j\}_{j \neq i}, d)$, to determine if there exists an isogeny $\phi : E \rightarrow E'$ of degree ℓ_i^d such that $\phi(P_j) = P'_j$ and $\phi(Q_j) = Q'_j$ for $j \neq i$.

Whilst we do not have an equivalence between the IND-Mask experiment and the *n*-*i*-DSIDH as presented above, we see that an oracle for the latter with $d = e_i$ is sufficient to obtain a noticeable advantage against the former. Also, it would seem that our IND-Mask experiment corresponds to a worst case of the *n*-*i*-DSIDH as it uses a maximal degree of $d = e_i$. Given the state of the art in cryptanalysis for these problems, we conjecture that the IND-Mask problem for \mathcal{M}_p is not significantly easier than the *n*-*i*-DISDH for the same parameters.

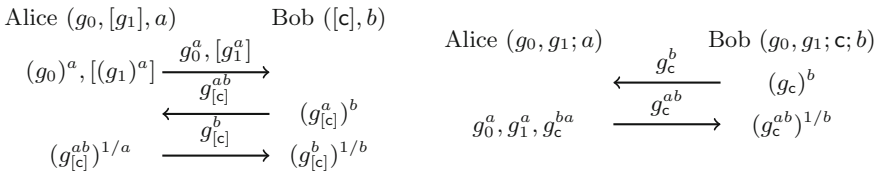
As hinted at in Note 1, the Weil pairing is in fact a useful tool against the IND-Mask experiment. Indeed, if the adversary had free control over the values r_0 and r_1 of the experiment, it could give two representatives whose basis points of the same torsion group evaluated to different values under the Weil pairing. This difference would be preserved under the secret masking action of the experiment and this would enable it to win trivially. Restricting the adversary’s input to be a single representative r and two masks that determine r_0 and r_1 and preserve the values of Weil pairing on the points of r thus prevents this strategy.

Security Analysis. As mentioned above, one of the main advantage of the SIDH approach as opposed to the hard homogeneous space approach (including CSIDH) is that no sub-exponential attack is known on the SIDH protocol, even using a quantum computer. On the other hand in the SIDH protocol, the action of the secret isogeny on a large torsion subgroup is revealed. A paper by Petit [33] and a recent follow-up work by Kutas et al. [29] show how to exploit this additional information to break variants of the SIDH protocol with unbalanced parameters or weak starting curves.

More precisely, let $N_1 \approx p^\alpha$ be the degree of the isogeny to compute, and let $N_2 \approx p^\beta$ be the order of torsion points images revealed in the protocol. The original SIDH protocol uses $\alpha \approx \beta \approx \frac{1}{2}$, but [33] and [29] describe a generalization to any coprime, power-smooth values N_1, N_2 . Under some parameter restrictions

and heuristic assumptions, the best attack in [29] computes the isogeny in classical polynomial time assuming $\beta > 2\alpha > 2$ or $\beta > 3\alpha > 3/2$. Furthermore, Kutas et al. show an attack requiring only $\beta > 2\alpha$ (with no lower bound on α) when the protocol uses a weak starting curve.

In our instantiation above, for any i one can fix $\alpha = e_i \log \ell_i$ and $\beta = \sum_{j \neq i} e_j \log \ell_j$. We also have $\alpha + \beta \leq 1$ so the first attack in [33] and its improvement in [29] does not apply if the starting curve is not weak. The second attack of [33], however, applies whenever the number n of factors ℓ_i is larger than $O(e_i \log \ell_i)$ for some i . The second one from [29] applies if any starting curve is weak. The notion of weak however depends on p , α , β and the chosen curve so choosing correct parameters (as those chosen in SIDH are) prevents this from happening.



(a) The Shamir three-pass protocol and its OT variant (b) Sketch of final OT protocol flows

Fig. 5. Sketch of the Shamir three-pass OT protocol and the final variant

One may fear that these attacks will get improved over time, leading to further restrictions on n . We note that $n = 3$ is sufficient to instantiate our OT protocols. Moreover, the protocol we describe in this paper could be even instantiated with $n = 2$. We note also that $n = 2$ in our construction corresponds to the SIDH protocol parameters, so our semi-commutative masking construction with $n = 2$ will remain secure as long as SIDH remains secure.

5 Oblivious Transfer Protocol from Masking Structures

In this section we construct an OT protocol from a semi-commutative masking structure \mathcal{M} . We prove its UC security for passive adversaries with static corruptions in the \mathcal{F}_{RO} -hybrid model assuming that \mathcal{M} is IND-Mask-secure and that the $\text{ParallelEither}^{\mathcal{M}}$ problem is hard.

Motivation. Our OT protocol is inspired by the two-party Shamir three-pass protocol for secure message transmission shown in Fig. 5a (ignoring the elements in square brackets), also known as the Massey-Omura encryption scheme. Here, Alice’s input is a message g together with a secret mask a and Bob’s input is another secret mask b . To transmit g , Alice first sends g^a to Bob who replies by

masking it as g^{ab} . Now Alice removes her mask and replies with $g^{ab/a} = g^b$ to Bob who then inverts b and recovers g . This protocol can be modified to yield an OT protocol by including the elements in square brackets; this was proposed by Wu et al. [40].

Alice, acting as Sender, now has two inputs g_0 and g_1 and masks both with a to send g_0^a, g_1^a to Bob, the Receiver. In addition to his mask b , Bob now also has a choice bit $c \in \{0, 1\}$ and he replies to Alice with $(g_c^a)^b$. They then continue as before until Bob recovers g_c . The intuition for security is that the mask a cannot be deduced from either g_0^a or g_1^a and therefore the first message hides both of Alice’s inputs from Bob. Also when Bob applies his own mask to one of the two messages, this hides his input bit c from Alice who does not know b .

Protocol Π_{OT}^1

PARAMETER: length n of the P_S ’s input strings.
 SENDER’S INPUT: $m_0, m_1 \in \mathcal{M}_E$.
 RECEIVER’S INPUT: $c \in \{0, 1\}$.
 COMMON INPUTS: Arbitrary $x_0 \neq x_1 \in X$ together with $r_0 \in R_{x_0}, r_1 \in R_{x_1}$ are shared and re-used for every instance of the protocol; an instance of the random oracle ideal functionality $\mathcal{F}_{RO} : \{0, 1\}^\lambda \rightarrow \mathcal{K}_E$.

OT₁ (Receiver 1)

- Sample $\beta \xleftarrow{\$} M_B$ uniformly at random.
- Compute $r_c^\beta := \beta(r_c)$ and $\beta^{-1} \in M_B$.
- Send r_c^β to P_S .

OT₂ (Sender 1)

- Sample $\alpha \xleftarrow{\$} M_A$ and compute $r_b^\alpha := \alpha(r_b) \in R_{x_b^\alpha}$, $b \in \{0, 1\}$
- For $b \in \{0, 1\}$, call \mathcal{F}_{RO} twice on input x_b^α obtaining k_b , and compute $e_b \leftarrow \text{Enc}(k_b, m_b)$
- Compute $r_c^{\alpha\beta} := \alpha(r_c^\beta)$
- Send $(r_c^{\alpha\beta}, e_0, e_1)$ to P_R .

OT₃ (Receiver 2)

- Compute $r_c^\alpha := \beta^{-1}(r_c^{\alpha\beta})$ and $k_R := \mathcal{F}_{RO}(x_c^\alpha)$ where $r_c^\alpha \in R_{x_c^\alpha}$.
- Return $m_c := \text{Dec}(k_R, e_c)$.

Fig. 6. The protocol Π_{OT}^1 for realizing \mathcal{F}_{OT} from semi-commutative masking.

We remove the need to apply the inverse mask $1/a$ to g_c^{ab} since Alice’s ignorance of c makes this impossible for general semi-commutative masking schemes due to the definition of inverse masks. In our new (discrete logarithm based) variant, the elements g_0 and g_1 are common to both parties. Rather than using a to send g_0^a, g_1^a to Bob (the Receiver), Alice (the Sender) does not go first. Instead, Bob first communicates his masked choice g_c^b , and then Alice applies her mask a and replies with g_c^{ab} . At that moment, she also computes g_0^a, g_1^a internally. She then uses these internal values to derive two symmetric keys k_0 and k_1 . Those

are used to encrypt Alice’s actual OT inputs m_0 and m_1 as two ciphertexts e_0 and e_1 which she sends alongside g_c^{ab} . This allows Bob to recover g_c^a and hence decrypt e_c to recover m_c . As g_0 and g_1 are now established once and re-used for every instance of the protocol, this allows the flows to have only *two* passes rather than three. Fig. 5b abstracts the symmetric encryption and only shows the flows that lead to Bob receiving the value g_c^a .

Construction. We now formally define our OT protocol from semi-commutative invertible masking schemes. Let $\mathcal{M} = \{X, R_X, [M_A, M_B, M_C]\}$ be an SCM structure with three masking sets; let $\mathcal{E} = \{(\text{KGen}_{\mathcal{E}}, \text{Enc}, \text{Dec}), (\mathcal{K}_{\mathcal{E}}, \mathcal{M}_{\mathcal{E}}, \mathcal{C}_{\mathcal{E}})\}$ be a symmetric encryption scheme and let \mathcal{F}_{RO} be an instance of the RO ideal functionality with domain $\mathcal{D} = X$ and range $\mathcal{R} = \mathcal{K}_{\mathcal{E}}$. The protocol Π_{OT}^1 is formally defined in Figure 6.

As described above, the idea of the protocol is that both the sender, P_S , and receiver, P_R , have as common input arbitrary elements $x_0 \neq x_1 \in X$ along with representations $r_0 \in R_{x_0}, r_1 \in R_{x_1}$. In the first pass, P_R takes a random mask $\beta \in M_B$ and sends $r_c^\beta = \beta(r_c)$ to P_S , where c is its choice bit. In the second pass, P_S samples a random mask $\alpha \in M_A$ and computes $r_0^\alpha = \alpha(r_0)$ and $r_1^\alpha = \alpha(r_1)$. These elements uniquely determine $x_b^\alpha \in X, b \in \{0, 1\}$. Thus the sender can compute two private keys $k_b, b \in \{0, 1\}$ (by invoking twice the random oracle functionality \mathcal{F}_{RO} on input x_b^α) and encrypt its input messages m_0, m_1 accordingly. P_S then sends the ciphertexts $e_b \leftarrow \text{Enc}(k_b, m_b), b \in \{0, 1\}$, and $r_c^{\alpha\beta} = \alpha(r_c^\beta)$ to P_R . The receiver has now all the information needed to recover the message m_c corresponding to its choice bit: it can apply the inverse β^{-1} to $r_c^{\alpha\beta}$ using the semi-commutativity of \mathcal{M} , so that

$$\beta^{-1}(r_c^{\alpha\beta}) = \beta^{-1}(\alpha(r_c^\beta)) = \beta^{-1}(\alpha(\beta(r_c))) \in R_{x_c^\alpha},$$

and recover $k_c = \mathcal{F}_{\text{RO}}(x_c^\alpha)$. This easily implies correctness of the scheme. Security is given by the following theorem. We give the proof in the full version and provide a sketch below.

Theorem 1. *The protocol Π_{OT}^1 of Fig. 6 securely UC-realizes the functionality \mathcal{F}_{OT} of Fig. 1 in the \mathcal{F}_{RO} -hybrid model for semi-honest adversaries and static corruptions, under the assumption that \mathcal{E} is IND-CPA-secure, that \mathcal{M} is IND-Mask-secure and that the ParallelEither ^{\mathcal{M}} problem is hard.*

Proof (sketch). We proceed by cases based on the honesty of each party. When both parties are corrupt, the simulator observes all the inputs and provides a perfect simulation. When only the receiver is corrupt, we build a reduction from a successful distinguishing environment first to the ParallelEither problem (by replacing k_{1-c} with a random one) and then to the IND-CPA security of \mathcal{E} (by replacing m_{1-c} with a random one). When only the sender is corrupt we build a reduction to the IND-Mask security of \mathcal{M} . When no party is corrupt, we combine the two previous reductions to simulate a protocol transcript without knowledge of c, m_0 and m_1 .

6 Active Secure Two-Round OT from Masking Structures

We now show how to compile our 2-round OT protocol Π_{OT}^1 , described in Sect. 5, to a 2-round maliciously UC-secure protocol using the generic transformations introduced by Döttling et al. [18].

6.1 Additional OT Security Notions

A 2-round OT protocol with public setup consists of four algorithms (Setup, $\text{OT}_1, \text{OT}_2, \text{OT}_3$) such that:

- Setup(1^λ) generates a public input pin.
- $\text{OT}_1(\text{pin}, c)$, where $c \in \{0, 1\}$ is the P_R choice bit, outputs (st, ot_{P_R})
- $\text{OT}_2(\text{pin}, \text{ot}_{P_R}, m_0, m_1)$, where m_0, m_1 are the sender’s input messages, outputs ot_{P_S}
- $\text{OT}_3(\text{st}, \text{ot}_{P_S})$ outputs m_c

First we need to recall some security notions [18] for the receiver P_R and the sender P_S . The first definition states that P_S should not learn anything about P_R ’s choice bit c .

Definition 11 (Receiver’s indistinguishability security). *For every PPT adversary \mathcal{A} :*

$$|\Pr[\mathcal{A}(\text{pin}, \text{OT}_1(\text{pin}, 0)) = 1] - \Pr[\mathcal{A}(\text{pin}, \text{OT}_1(\text{pin}, 1)) = 1]| = \text{negl}(\lambda),$$

where pin is the public output of the setup phase.

The next definition concerns the security of the sender; it states that P_R cannot compute both secret values y_0 and y_1 used by OT_2 to protect m_0 and m_1 , but not necessarily in the same experiment.

Definition 12 (Sender’s search security). *Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary where \mathcal{A}_2 outputs a string y^* . Consider the following experiment $\text{Exp}_{\text{sOT}}^{\text{pin}, \rho, w}(\mathcal{A})$, indexed by a pin, random coins $\rho \in \{0, 1\}^\lambda$ and a bit $w \in \{0, 1\}$.*

1. Run $(\text{ot}_{P_R}, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda, \text{pin}; \rho)$.
2. Compute $(\text{ot}_{P_S}, y_0, y_1) \stackrel{\$}{\leftarrow} \text{OT}_2(\text{pin}, \text{ot}_{P_R})$.
3. Run $y^* \leftarrow \mathcal{A}_2(\text{st}, \text{ot}_{P_S}, w)$ and output 1 iff $y^* = y_w$.

We say that \mathcal{A} breaks a scheme’s Sender’s search (sOT) security if there exists a non-negligible function ϵ such that

$$\Pr_{\text{pin}, \rho} [\Pr[\text{Exp}_{\text{sOT}}^{\text{pin}, \rho, 0}(\mathcal{A}) = 1] > \epsilon \text{ and } \Pr[\text{Exp}_{\text{sOT}}^{\text{pin}, \rho, 1}(\mathcal{A}) = 1] > \epsilon] > \epsilon,$$

where $\text{pin} \stackrel{\$}{\leftarrow} \text{Setup}$ and $\rho \stackrel{\$}{\leftarrow} \{0, 1\}^\lambda$.

6.2 Two Rounds OT with Active UC-Security

We provide an intermediary result which enables us to use the general compiler from [18] to get an actively secure 2-round OT protocol starting from Π_{OT}^1 . First we introduce and discuss a new security assumption derived from the Parallel problem but more suited to active adversaries. Then we show that our protocol satisfies the security notions of Definitions 11 and 12. Finally, by applying the general transformations from sOT to UC OT described in [18], we obtain a fully UC-secure two-round OT protocol. We note that we are able to remove the random oracle from our protocol to achieve sOT security; therefore the resulting OT protocol requires only the CRS. We define our new computational problem as follows.

Definition 13 (ParallelDouble). *Given $(i, j, r, r_{x_0}, r_{x_1}, r_y)$ with the promise that $i \neq j$ and that $r_{x_b} = \mu_{x_b}(r)$, $b \in \{0, 1\}$ and $r_y = \mu_y(r)$ for random $\mu_{x_b} \xleftarrow{\$} M_i$ and $\mu_y \xleftarrow{\$} M_j$, and given a one-time access to an oracle \mathcal{O}_y which, when given $r \in R$ returns $\mu_y(r)$, compute $z_0, z_1 \in X$ such that both $\mu_{x_b}(r_y) \in R_{z_b}$.*

The instantiation of this problem in the discrete logarithm case is, when given (g, g^a, g^b, g^c) and a one-time access to an exponentiation-by- c oracle, to return both g^{ac} and g^{bc} . For practical efficiency, it is also desirable that g^a and g^b remain constant across multiple instances of the ParallelDouble problem, with only g^c being randomly sampled in each instance. This version of the problem is similar to the one-more static CDH problem where an adversary has to successfully compute one more CDH challenge than it was able to ask from a helper oracle [9].

Security of the Π_{OT}^1 protocol. We then prove that protocol Π_{OT}^1 achieves Receiver’s indistinguishability and Sender’s search security.

Proposition 1. *The protocol Π_{OT}^1 in Fig. 6 satisfies computational receiver’s indistinguishability security and sender’s sOT security under the assumption that \mathcal{M} is IND-Mask-secure and that the ParallelDouble ^{\mathcal{M}} problem is hard.*

Proof. Receiver’s indistinguishability follows from the IND-Mask-security assumption. By setting the public inputs r_0 and r_1 in Π_{OT}^1 as they are computed in the IND-Mask experiment, the random mask μ is distributed in the same way as the mask β in OT_1 . Therefore if an adversary breaks the receiver’s indistinguishability for Π_{OT}^1 , this can be reduced to a solution to the IND-Mask problem.

Sender’s Search Security. To prove sOT security for Π_{OT}^1 we assume the existence of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and a non-negligible ϵ such that

$$\Pr_{\text{pin}, \rho} [\Pr[\text{Exp}_{\text{sOT}}^{\text{pin}, \rho, 0}(\mathcal{A}) = 1] > \epsilon \text{ and } \Pr[\text{Exp}_{\text{sOT}}^{\text{pin}, \rho, 1}(\mathcal{A}) = 1] > \epsilon] > \epsilon,$$

and we build a reduction \mathcal{B} that is given a ParallelDouble challenge $(i, j, r, r_{x_0}, r_{x_1}, r_y)$ with access to an oracle \mathcal{O}_y (Definition 13). Instead of running Setup to

generate r_0 and r_1 , \mathcal{B} sets $r_0 \leftarrow r_{x_0}$ and $r_1 \leftarrow r_{x_1}$; also \mathcal{B} samples $\rho \xleftarrow{\$} \{0, 1\}^\lambda$. As this ensures that pin is distributed identically to the output of Setup , pin and ρ are good for \mathcal{A} with probability at least ϵ .

After \mathcal{B} runs \mathcal{A}_1 , which outputs $(\text{ot_}P_R, \text{st})$, it queries the oracle to obtain $\text{ot_}P_{S,0} \leftarrow \mathcal{O}_y(\text{ot_}P_R)$. It also computes $\text{ot_}P_{S,1} \leftarrow \mu(\text{ot_}P_{S,0})$ for a random $\mu \in M_k$ with $i \neq k \neq j$; it also computes μ^{-1} . Then, for $w \in \{0, 1\}$, \mathcal{B} runs $y_w^* \leftarrow \mathcal{A}_2(\text{st}, \text{ot_}P_{S,w}, w)$ and updates $y_1^* \leftarrow \mu^{-1}(y_1^*)$. Finally \mathcal{B} returns y_0^* and the updated y_1^* as the ParallelDouble answer.

Since $\Pr[\text{Exp}_{\text{OT}}^{\text{pin}, \rho, 0}(\mathcal{A}) = 1] > \epsilon$ and $\Pr[\text{Exp}_{\text{OT}}^{\text{pin}, \rho, 1}(\mathcal{A}) = 1] > \epsilon$, with probability ϵ^2 , \mathcal{A}_2 is successful for both inputs $(\text{st}, \text{ot_}P_{S,0}, 0)$ and $(\text{st}, \text{ot_}P_{S,1}, 1)$ as the two messages are made independent by \mathcal{B} 's addition of μ . If this happens, then y_0^* is exactly one of the answers, and the update of y_1^* by \mathcal{B} removes the extra mask μ and means that y_1^* is then the other answer to the ParallelDouble problem. Hence \mathcal{B} is successful with probability at least ϵ^3 .

Theorem 2. *Under the assumption that \mathcal{M} is IND-Mask-secure and that the $\text{ParallelDouble}^{\mathcal{M}}$ problem is hard, there exists a 2-round UC-secure OT protocol constructed from Π_{OT}^1 .*

Proof. This follows from the transformations and results of [18, Theorems 8, 9, 11, 12, 14, 19 and 21].

Corollary 1. *By instantiating the semi-commutative masking scheme, there exists an actively secure 2-round OT protocol based on supersingular isogenies.*

We note that the isogeny-based OT protocols proposed by Vitse [38] requires three rounds of communication; this implies that they cannot be transformed to achieve two-round OT with fully UC-security using this transformation.

Acknowledgements. This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPACT, by CyberSecurity Research Flanders with reference number VR20192203, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contracts No. N66001-15-C-4070 and No. HR001120C0085, by the FWO under an Odysseus project GOH9718N and by EPSRC grant EP/S01361X/1

References

1. Adj, G., Ahmadi, O., Menezes, A.: ON isogeny graphs of supersingular elliptic curves over finite fields. Cryptology ePrint Archive, Report 2018/132 (2018). <https://eprint.iacr.org/2018/132>
2. Azarderakhsh, R., Jalali, A., Jao, D., Soukharev, V.: Practical supersingular isogeny group key agreement. Cryptology ePrint Archive, Report 2019/330 (2019), <https://eprint.iacr.org/2019/330>
3. Azarderakhsh, R., Jao, D., Kalach, K., Koziel, B., Leonardi, C.: Key compression for isogeny-based cryptosystems. In: Emura, K., Hanaoka, G., Zhang, R. (eds.) Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, APKC, pp. 1–10. ACM (2016)

4. Barreto, P., Oliveira, G., Benits, W.: Supersingular isogeny oblivious transfer. Cryptology ePrint Archive, Report 2018/459 (2018). <https://eprint.iacr.org/2018/459>
5. Barreto, P.S.L.M., David, B., Dowsley, R., Morozov, K., Nascimento, A.C.A.: A framework for efficient adaptively secure composable oblivious transfer in the ROM. Cryptology ePrint Archive, Report 2017/993 (2017). <http://eprint.iacr.org/2017/993>
6. Brakerski, Z., Döttling, N.: Two-message statistically sender-private OT from LWE. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018. LNCS, vol. 11240, pp. 370–390. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03810-6_14
7. Branco, P., Ding, J., Goulão, M., Mateus, P.: A framework for universally composable oblivious transfer from one-round key-exchange. In: Albrecht, M. (ed.) IMACC 2019. LNCS, vol. 11929, pp. 78–101. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35199-1_5
8. Branco, P., Ding, J., Goulão, M., Mateus, P.: A framework for universally composable oblivious transfer from one-round key-exchange. Cryptology ePrint Archive, Report 2019/726 (2019). <https://eprint.iacr.org/2019/726>. To appear at the 17th IMA International Conference on Cryptography and Coding
9. Bresson, E., Monnerat, J., Vergnaud, D.: Separation results on the “one-more” computational problems. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 71–87. Springer, Heidelberg (Apr (2008)
10. Byali, M., Patra, A., Ravi, D., Sarkar, P.: Fast and universally-composable oblivious transfer and commitment scheme with adaptive security. Cryptology ePrint Archive, Report 2017/1165 (2017). <https://eprint.iacr.org/2017/1165>
11. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001
12. 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
13. Childs, A., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum subexponential time. *J. Math. Cryptol.* **8**(1), 1–29 (2014). a pre-print version appears at <https://arxiv.org/abs/1012.4019>
14. Chou, T., Orlandi, C.: The simplest protocol for oblivious transfer. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 40–58. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22174-8_3
15. 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
16. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006). <http://eprint.iacr.org/2006/291>
17. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.* **8**(3), 209–247 (2014). a pre-print version appears at <https://eprint.iacr.org/2011/506>
18. Döttling, N., Garg, S., Hajiabadi, M., Masny, D., Wichs, D.: Two-round oblivious transfer from CDH or LPN. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 768–797. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_26

19. Faz-Hernández, A., López, J., Ochoa-Jiménez, E., Rodríguez-Henríquez, F.: A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. *IEEE Trans. Comput.* **67**(11), 1622–1636 (2018)
20. Fujioka, A., Takashima, K., Terada, S., Yoneyama, K.: Supersingular isogeny Diffie-Hellman authenticated key exchange. In: Lee, K. (ed.) *ICISC 18. LNCS*, vol. 11396, pp. 177–195. Springer, Heidelberg (Nov (2019)
21. Fujioka, A., Takashima, K., Yoneyama, K.: One-round authenticated group key exchange from isogenies. In: Steinfeld, R., Yuen, T.H. (eds.) *ProvSec 2019. LNCS*, vol. 11821, pp. 330–338. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31919-9_20
22. Galbraith, S.: Isogeny crypto. Blog post from Ellipticnews (2019). <https://ellipticnews.wordpress.com/2019/11/09/isogeny-crypto/>. Accessed 15 Apr 2020
23. Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: Cheon, J.H., Takagi, T. (eds.) *ASIACRYPT 2016. LNCS*, vol. 10031, pp. 63–91. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_3
24. Galbraith, S.D., Petit, C., Silva, J.: Identification protocols and signature schemes based on supersingular isogeny problems. In: Takagi, T., Peyrin, T. (eds.) *ASIACRYPT 2017. LNCS*, vol. 10624, pp. 3–33. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_1
25. Galbraith, S.D., Vercauteren, F.: Computational problems in supersingular elliptic curve isogenies. *Quant. Inf. Process.* **17**(10), 1–22 (2018). <https://doi.org/10.1007/s11128-018-2023-6>
26. Goldreich, O., Oren, Y.: Definitions and properties of zero-knowledge proof systems. *J. Cryptol.* **7**(1), 1–32 (1994). <https://doi.org/10.1007/BF00195207>
27. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.Y. (ed.) *Post-Quantum Cryptography*, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
28. Keller, M., Orsini, E., Scholl, P.: MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) *ACM CCS 2016*, pp. 830–842. ACM Press, October 2016
29. Kutas, P., Martindale, C., Panny, L., Petit, C., Stange, K.E.: Weak instances of sidh variants under improved torsion-point attacks. *Cryptology ePrint Archive, Report 2020/633* (2020). <https://eprint.iacr.org/2020/633>
30. Lai, Y.F., Galbraith, S.D., Delpech de Saint Guilhem, C.: Compact, efficient and uc-secure isogeny-based oblivious transfer. *Cryptology ePrint Archive, Report 2020/1012* (2020). <https://eprint.iacr.org/2020/1012>
31. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) *CRYPTO 2012. LNCS*, vol. 7417, pp. 681–700. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_40
32. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) *CRYPTO 2008. LNCS*, vol. 5157, pp. 554–571. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_31
33. Petit, C.: Faster algorithms for isogeny problems using torsion point images. In: Takagi, T., Peyrin, T. (eds.) *ASIACRYPT 2017. LNCS*, vol. 10625, pp. 330–353. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_12
34. Rabin, M.O.: How to exchange secrets by oblivious transfer. Technical report TR-81, Aiken Computation Laboratory, Harvard University (1981)

35. Sahu, R.A., Gini, A., Pal, A.: Supersingular isogeny-based designated verifier blind signature. Cryptology ePrint Archive, Report 2019/1498 (2019). <https://eprint.iacr.org/2019/1498>
36. Silverman, J.H.: The arithmetic of elliptic curves, Graduate Texts in Mathematics, vol. 106. Springer, New York (1986). <https://doi.org/10.1007/978-1-4757-1920-8>
37. Urbanick, D., Jao, D.: Sok: the problem landscape of sidh. In: APKC 2018: Proceedings of the 5th ACM on ASIA Public-Key Cryptography Workshop, pp. 53–60. ACM (2018)
38. Vitse, V.: Simple oblivious transfer protocols compatible with supersingular isogenies. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2019. LNCS, vol. 11627, pp. 56–78. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23696-0_4
39. Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 39–56. ACM Press, October/November 2017
40. Wu, Q.-H., Zhang, J.-H., Wang, Y.-M.: Practical t-out-n oblivious transfer and its applications. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 226–237. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39927-8_21



Optimized and Secure Pairing-Friendly Elliptic Curves Suitable for One Layer Proof Composition

Youssef El Housni^{1,2,3}  and Aurore Guillevic⁴ 

¹ EY Blockchain, Paris, France

`youssef.el.housni@fr.ey.com`

² LIX, CNRS, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France

³ Inria, Palaiseau, France

⁴ Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
`aurore.guillevic@inria.fr`

Abstract. A zero-knowledge proof is a method by which one can prove knowledge of general non-deterministic polynomial (NP) statements. SNARKs are in addition non-interactive, short and cheap to verify. This property makes them suitable for recursive proof composition, that is proofs attesting to the validity of other proofs. To achieve this, one *moves* the arithmetic operations *to the exponents*. Recursive proof composition has been empirically demonstrated for pairing-based SNARKs via tailored constructions of expensive pairing-friendly elliptic curves namely a pair of 753-bit MNT curves, so that one curve's order is the other curve's base field order and vice-versa. The ZEXE construction restricts to one layer proof composition and uses a pair of curves, BLS12-377 and CP6-782, which improve significantly the arithmetic on the first curve. In this work we construct a new pairing-friendly elliptic curve to be used with BLS12-377, which is STNFS-secure and fully optimized for one layer composition. We propose to name the new curve BW6-761. This work shows that it is at least five times faster to verify a composed SNARK proof on this curve compared to the previous state-of-the-art, and proposes an optimized Rust implementation that is almost thirty times faster than the one available in ZEXE library.

1 Introduction

Proofs of knowledge are a powerful tool introduced in [19] and studied both in theoretical and applied cryptography. Since then, cryptographers designed and improved short, non-interactive and cheap to verify proofs, resulting in Succinct Non-interactive ARGuments of Knowledge (SNARKs). Zero-knowledge (zk) SNARKs allow a prover to convince a verifier that they know a witness to an

Full version available on eprint at <https://eprint.iacr.org/2020/351>.

© Springer Nature Switzerland AG 2020

S. Krenn et al. (Eds.): CANS 2020, LNCS 12579, pp. 259–279, 2020.

https://doi.org/10.1007/978-3-030-65411-5_13

instance, which is a member of a language in NP, whilst revealing no information about this witness. The verification of the proof should be fast. The discrete logarithm problem: given a finite cyclic group \mathbb{G} , a generator g , and $h \in \mathbb{G}$, find $x = \log_g h$ such that $h = g^x$, is at the heart of many proofs of knowledge. Making a scheme non-interactive leads to a signature scheme, such as Schnorr protocol and signature.

A cryptographic bilinear pairing is a map $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ where \mathbb{G}_1 and \mathbb{G}_2 are distinct subgroups of an elliptic curve defined over a finite field \mathbb{F}_q , and \mathbb{G}_T is an extension field \mathbb{F}_{q^k} . Pairings allow to multiply hidden values in the exponents: with a multiplicative notation for the three groups, and generators g_1, g_2 for $\mathbb{G}_1, \mathbb{G}_2$, one has $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$: a pairing can multiply two discrete logarithms without revealing them. As of 2020, the most efficient proof-of-knowledge scheme due to Groth [20] is a pre-processing zk-SNARK made of bilinear pairings. Hence, constructions of pairing-friendly elliptic curves are required.

Besides efficiency, SNARKs' succinctness makes them good candidates for recursive proof composition. Such proofs could themselves verify the correctness of (a batch of) other proofs. This would allow a single proof to inductively attest to the correctness of many former proofs. However, once a first proof is generated, it is highly impractical to use the same elliptic curve to generate a second proof verifying the first one. A practical approach requires two different curves that are closely tied together. Therefore, we need tailored pairing-friendly curves that are usually expensive to construct and to use.

1.1 Previous Work

Ben-Sasson et al. [4] presented the first practical setting of recursive proof composition with a cycle of two MNT pairing-friendly elliptic curves [16, Sec. 5]. Proofs generated from one curve can feasibly reason about proofs generated from the other curve. To achieve this, one curve's order is the other curve's base field order and vice-versa. But, both are quite expensive at the 128-bit security level. The two curves have low embedding degrees (4 and 6) resulting in large base fields to achieve a standard security level. For example, the Coda protocol [26] implements curves of 753 bits. Moreover, Chiesa et al. [12] established some limitations on finding other suitable cycles of curves.

On the other hand, Bowe et al. proposed the Zexe system [7]. They use a relatively relaxed approach to find a suitable pair of curves that forms a chain rather than a cycle. The authors constructed a BLS12 curve to generate the inner proofs while allowing the construction of a second curve via the Cocks–Pinch method [16, Sec. 4.1] to generate the outer proof. It is to note that while the inner curve is efficient at 128-bit security level, the outer curve is quite expensive.

1.2 Our Contributions

We present a new secure and optimized pairing-friendly elliptic curve suitable for one-layer proof composition and much faster than the previous state-of-the-art.

To achieve this, we moved from the Cocks–Pinch to the Brezing–Weng method to generate curves. Our curve can substitute for Zexe’s outer curve while enjoying a very efficient implementation. The curve is defined over a 761-bit prime field instead of 782 bits, saving one machine-word of 64 bits. The curve has CM discriminant $-D = -3$, allowing fast GLV scalar multiplication on \mathbb{G}_1 and \mathbb{G}_2 . The curve has embedding degree 6 and a twist of degree 6, and \mathbb{G}_2 has coordinates in the same prime field as \mathbb{G}_1 (factor 6 compression). The curve also has fast subgroup checking and fast cofactor multiplication. Finally, we obtain a very efficient optimal ate pairing on this curve.

In particular, we show it is at least five times faster to verify a Groth proof, compared to Zexe. We provide an optimized Rust implementation that achieves a speedup factor of almost 30.

1.3 Applications

We briefly mention some applications from the blockchain community projects that can benefit from this work:

- Zexe** The authors introduced the notion of Decentralized Private Computation (DPC) that uses one layer proof composition [7]. As an application, they described user-defined assets, decentralized exchanges and policy-enforcing stablecoins in [7, § V].
- Celo** The project aims to develop a mobile-first oriented blockchain platform. Celo verifies BLS signatures by generating a single SNARK proof that verifies many signatures [9].
- EY Blockchain** The firm released its Nightfall tool [14] into the public domain, a smart-contract based solution leveraging zkSNARKs for private transactions of fungible and non-fungible tokens on the Ethereum blockchain. Recently, EY unveiled its latest Nightfall upgrade allowing for transaction batching. This work can be used to aggregate many Nightfall proofs into one, thus reducing the overall gas cost.
- Filecoin** The protocol [27] describes a decentralized storage blockchain. Protocol Labs introduced Proof-of-Replication that can be used to prove that some data has been replicated to its own uniquely dedicated physical storage. This proof is then compressed using a SNARK proof but this results in a massive arithmetic circuit. Filecoin is considering splitting the circuit into 20 smaller ones and generating small proofs that can be aggregated into one using one layer proof composition.

Organization of the Paper. In Sect. 2, we provide preliminaries on pairing-friendly elliptic curves and recursive proof composition. In Sect. 3, we introduce our curve, discuss the optimizations and compare it to Zexe’s outer curve. We estimate in Sect. 4 the security of Zexe’s inner curve and our curve, taking into account the Special Tower NFS algorithm, and Cheon’s attack. We conclude in Sect. 5.

2 Preliminaries

We present the background on pairing-friendly elliptic curves and recursive composition of zk-SNARKs proofs that is needed to understand our curve’s construction.

2.1 Pairing-Friendly Elliptic Curves

Background on Pairings. We briefly recall elementary definitions on pairings and present the computation of two pairings used in practice, the Tate and ate pairings. All elliptic curves discussed below are *ordinary* (i.e. non-supersingular).

Let E be an elliptic curve defined over a field \mathbb{F}_q , where q is a prime power. Let π_q be the Frobenius endomorphism: $(x, y) \mapsto (x^q, y^q)$. Its minimal polynomial is $X^2 - tX + q$ where t is called the *trace*. Let r be a prime divisor of the curve order $\#E(\mathbb{F}_q) = q + 1 - t$. The r -torsion subgroup of E is denoted $E[r] := \{P \in E(\mathbb{F}_q), [r]P = \mathcal{O}\}$ and has two subgroups of order r (eigenspaces of ϕ_q in $E[r]$) that are useful for pairing applications. We define the two groups $\mathbb{G}_1 = E[r] \cap \ker(\pi_q - [1])$ with a generator denoted by G_1 , and $\mathbb{G}_2 = E[r] \cap \ker(\pi_q - [q])$ with a generator G_2 . The group \mathbb{G}_2 is defined over \mathbb{F}_{q^k} , where the embedding degree k is the smallest integer $k \in \mathbb{N}^*$ such that $r \mid q^k - 1$.

We recall the Tate and ate pairing definitions, based on the same two steps: evaluating a function $f_{s,Q}$ at a point P , the Miller loop step, and then raising it to the power $(q^k - 1)/r$, the final exponentiation step. The function $f_{s,Q}$ has divisor $\text{div}(f_{s,Q}) = s(Q) - ([s]Q) - (s - 1)(\mathcal{O})$ and satisfies, for integers i and j ,

$$f_{i+j,Q} = f_{i,Q} f_{j,Q} \frac{\ell_{[i]Q, [j]Q}}{v_{[i+j]Q}},$$

where $\ell_{[i]Q, [j]Q}$ and $v_{[i+j]Q}$ are the two lines needed to compute $[i+j]Q$ from $[i]Q$ and $[j]Q$ (ℓ intersecting the two points and v the vertical). We compute $f_{s,Q}(P)$ with the Miller loop presented in Algorithm 1.

Algorithm 1: MillerLoop(s, P, Q)

```

Output:  $m = f_{s,Q}(P)$ 
1  $m \leftarrow 1; S \leftarrow Q;$ 
2 for  $b$  from the second most significant bit of  $s$  to the least do
3    $\ell \leftarrow \ell_{S,S}(P); S \leftarrow [2]S;$  DOUBLELINE
4    $v \leftarrow v_{[2]S}(P);$  VERTICALLINE
5    $m \leftarrow m^2 \cdot \ell/v;$  UPDATE1
6   if  $b = 1$  then
7      $\ell \leftarrow \ell_{S,Q}(P); S \leftarrow S + Q;$  ADDLINE
8      $v \leftarrow v_{S+Q}(P);$  VERTICALLINE
9      $m \leftarrow m \cdot \ell/v;$  UPDATE2
10 return  $m;$ 

```

Algorithm 2: Cocks–Pinch method

- Input:** A positive integer k and a positive square-free integer D
Output: E/\mathbb{F}_q with an order- r subgroup and embedding degree k
- 1 Fix k and D and choose a prime r such that k divides $r - 1$ and $-D$ is a square modulo r ;
 - 2 Compute $t = 1 + x^{(r-1)/k}$ for x a generator of $(\mathbb{Z}/r\mathbb{Z})^\times$;
 - 3 Compute $y = (t - 2)/\sqrt{-D} \pmod r$;
 - 4 Lift t and y in \mathbb{Z} ;
 - 5 Compute $q = (t^2 + Dy^2)/4$ in \mathbb{Q} ;
 - 6 **if** q is a prime integer **then**
 - 7 | Use CM method ($D < 10^{12}$) to construct E/\mathbb{F}_q with order- r subgroup;
 - 8 **else**
 - 9 | Go back to 1;
 - 10 **return** E/\mathbb{F}_q with an order- r subgroup and embedding degree k

The Tate and ate pairings are defined by

$$\begin{aligned} \text{Tate}(P, Q) &:= f_{r,P}(Q)^{(q^k-1)/r} \\ \text{ate}(P, Q) &:= f_{t-1,Q}(P)^{(q^k-1)/r} \end{aligned}$$

where $P \in \mathbb{G}_1 \subset E[r](\mathbb{F}_q)$ and $Q \in \mathbb{G}_2 \subset E[r](\mathbb{F}_{q^k})$. The values $\text{Tate}(P, Q)$ and $\text{ate}(P, Q)$ are in the *target* group \mathbb{G}_T of r -th roots of unity in \mathbb{F}_{q^k} . In this paper, when abstraction is needed, we denote a pairing as follows $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.

It is also important to recall some results with respect to the complex multiplication (CM) discriminant $-D$. When $D = 3$ (resp. $D = 4$), the curve has CM by $\mathbb{Q}(\sqrt{-3})$ (resp. $\mathbb{Q}(\sqrt{-1})$) so that twists of degrees 3 and 6 exist (resp. 4). When E has d -th order twists for some $d \mid k$, then \mathbb{G}_2 is isomorphic to $E'[r](\mathbb{F}_{q^{k/a}})$ for some twist E' . Otherwise, in the general case, E admits a single twist (up to isomorphism) and it is of degree 2.

Some Pairing-Friendly Constructions. Here, we recall some methods from the literature for constructing pairing-friendly ordinary elliptic curves that will be of interest in the following sections. We focus on the Cocks–Pinch [16, Sec. 4.1], Barreto–Lynn–Scott [16, Sec. 6.1] and Brezing–Weng [16, Sec. 6.1] methods, but also mention the Miyaji–Nakabayashi–Takano (MNT) curves [16, Sec. 5.1].

Cocks–Pinch is the most flexible of the above methods and can be used to construct curves with arbitrary embedding degrees but with ratio $\rho = \log_2 q / \log_2 r \approx 2$. It works by fixing the subgroup order r and the CM discriminant D and then computing the trace t and the prime q such that the CM equation $4q = t^2 + Dy^2$ (for some $y \in \mathbb{Z}$) is satisfied (cf. Algorithm 2).

Brezing and Weng [16, Sec. 6.1], and independently, Barreto, Lynn and Scott [16, Sec. 6.1] generalized the Cocks–Pinch method by parametrizing t, r and q as polynomials. This led to curves with ratio $\rho < 2$. Below, we sketch the idea of the algorithm in its generality for both BLS and BW constructions (cf. Algorithm 3).

Algorithm 3: Idea of BLS and BW methods

- Input:** A positive integer k and a positive square-free integer D
Output: $E/\mathbb{F}_{q(x)}$ with an order- $r(x)$ subgroup and embedding degree k
- 1 Fix k and D and choose an irreducible polynomial $r(x) \in \mathbb{Z}[x]$ with positive leading coefficient¹ such that $\sqrt{-D}$ and the primitive k -th root of unity ζ_k are in $K = \mathbb{Q}[x]/(r(x))$;
 - 2 Choose $t(x) \in \mathbb{Q}[x]$ be a polynomial representing $\zeta_k + 1$ in K ;
 - 3 Set $y(x) \in \mathbb{Q}[x]$ be a polynomial mapping to $(\zeta_k - 1)/\sqrt{-D}$ in K ;
 - 4 Compute $q(x) = (t^2(x) + Dy^2(x))/4$ in $\mathbb{Q}[x]$;
 - 5 **return** $E/\mathbb{F}_{q(x)}$ with an order- $r(x)$ subgroup and embedding degree k
-

¹so that $r(x)$ satisfies Bunyakovsky conjecture, which states that such a polynomial produces infinitely many primes for infinitely many integers.

Table 1. Polynomial parameters of BLS12 curve family.

BLS12, $k = 12$, $D = 3$, $x = 1 \pmod 3$
$q_{\text{BLS12}}(x) = (x^6 - 2x^5 + 2x^3 + x + 1)/3$, $x = 1 \pmod 3$
$r_{\text{BLS12}}(x) = x^4 - x^2 + 1$
$t_{\text{BLS12}}(x) = x + 1$

A particular choice of polynomials for $k = 12$ yields a family of curves with a good security/performance tradeoff, denoted BLS12. The parameters are given in Table 1. MNT curves, however, have a fixed embedding degree $k \in \{3, 4, 6\}$ and variable discriminant D . For $k = 6$, one has $p_6(x) = 4x^2 + 1$, $r_6(x) = 4x^2 - 2x + 1$, and for $k = 4$, $p_4(x) = x^2 + x + 1$, $r_4(x) = x^2 + 1$. One has $p_6(x) = r_4(-2x)$ and $r_6(x) = p_4(-2x)$. If p_6, r_6, p_4, r_4 are prime, this makes a cycle of pairing-friendly curves.

Pairing-Friendly Chains and Cycles. Two elliptic curves defined over finite fields form a *chain* if the characteristic of the field of definition of one curve equals the number of points on the next. If this property is cyclic, then it is called a *cycle*. These concepts are generalized to m curves by the following definitions.

Definition 1. An m -chain of elliptic curves is a list of distinct curves

$$E_1/\mathbb{F}_{q_1}, \dots, E_m/\mathbb{F}_{q_m}$$

where q_1, \dots, q_m are large primes and

$$\#E_2(\mathbb{F}_{q_2}) = q_1, \dots, \#E_i(\mathbb{F}_{q_i}) = q_{i-1}, \dots, \#E_m(\mathbb{F}_{q_m}) = q_{m-1} . \tag{1}$$

Definition 2. An m -cycle of elliptic curves is an m -chain, with

$$\#E_1(\mathbb{F}_{q_1}) = q_m . \tag{2}$$

In the literature, a 2-cycle of ordinary curves is called an *amicable pair*. Following the same logic, we call a 2-chain of pairing-friendly ordinary elliptic curves *pairing-friendly amicable chain*. In this paper, we are interested in constructing a pairing-friendly amicable chain of curves with efficient arithmetic.

2.2 Recursive Proof Composition

To date the most efficient zkSNARK is due to Groth [20]. Here, we briefly sketch its construction and refer the reader to the original paper. The construction consists of a trapdoored setup and a proof made of 3 group elements. The verification is one equation of a pairing product (Eq. 3), for a pairing $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order r defined over a field \mathbb{F}_q . Given an instance $\Phi = (a_0, \dots, a_l) \in \mathbb{F}_r^l$, a proof $\pi = (A, C, B) \in \mathbb{G}_1^2 \times \mathbb{G}_2$ and a verification key $vk = (vk_{\alpha, \beta}, \{vk_{\pi_i}\}_{i=0}^l, vk_\gamma, vk_\delta) \in \mathbb{G}_T \times \mathbb{G}_1^{l+1} \times \mathbb{G}_2^2$, the verifier must check that

$$e(A, B) = vk_{\alpha, \beta} \cdot e(vk_x, vk_\gamma) \cdot e(C, vk_\delta), \tag{3}$$

where $vk_x = \sum_{i=0}^l [a_i]vk_{\pi_i}$ depends only on the instance Φ and $vk_{\alpha, \beta} = e(vk_\alpha, vk_\beta)$ can be precomputed in the trusted setup for $(vk_\alpha, vk_\beta) \in \mathbb{G}_1 \times \mathbb{G}_2$.

Note that, for an efficient implementation, the subgroup order r is chosen to allow efficient FFT-based polynomial multiplications, as proposed in [3]. To achieve this, we require *high 2-adicity*: $r - 1$ should be divisible by a *large enough* power of 2.

To allow recursive proof composition, one needs to write the verification Eq. (3) as an instance in the prover language. In pairing-based SNARKs such as [20], the verification arithmetic (computing the scalar multiplications and the pairings of (3)) is in an extension of \mathbb{F}_q up to a degree k while the proving arithmetic is in \mathbb{F}_r . That means we need another pairing $e': \mathbb{G}'_1 \times \mathbb{G}'_2 \rightarrow \mathbb{G}'_T$ where the groups \mathbb{G}'_i are of prime order q (instead of r) to prove the arithmetic operations over \mathbb{F}_q . Since a pairing-friendly curve with $q = r$ doesn't exist¹, one would need to simulate \mathbb{F}_q operations via \mathbb{F}_r operations which results in a computational blowup of order $\log q$ compared to native arithmetic.

A practical alternative was suggested in [4] using a pairing-friendly amicable pair. The authors proposed a cycle of two MNT curves [16, Sec.5], E_4 and E_6 , with embedding degrees 4 and 6 and primes q and r of 298 bits. One has $\#E_4(\mathbb{F}_r) = q$ and $\#E_6(\mathbb{F}_q) = r$. While this solves our problem, the security level of the curves is *low*. To remediate this, the Coda protocol [26] uses a larger MNT-based amicable pair proposed by [4] that targets 128-bit security with primes q, r of 753 bits, but at the cost of expensive computations (cf. Fig. 1).

While an amicable pair allows an infinite recursion loop, an amicable chain allows a bounded recursion. In some applications, such as those we mentioned, a one-layer composition is sufficient. To this end, Zexe's authors proposed an amicable chain consisting of an inner BLS12 curve called BLS12-377 and an

¹ r needs to divide $q^k - 1$ for $k \in \mathbb{N}^* [1, 20]$, thus $r = 1$ is the only solution.

outer Cocks–Pinch curve called CP6-782 (for Cocks–Pinch method, embedding degree 6 and field size of 782 bits). BLS12-377 was constructed to have both $r - 1$ and $q - 1$ highly 2-adic while enjoying all the efficient implementation properties of the BLS12 family. With the inner curve constructed, the authors looked for a pairing-friendly curve with pre-determined subgroup order r equal to the field size q of BLS12-377. The only construction from the literature to allow such flexibility is Cocks–Pinch, but it unfortunately results in a curve on which operations are at least two times more costly (in time and space) than BLS12-377. Furthermore, this outer curve CP6-782 doesn't allow efficient pairing computation and efficient scalar multiplication via endomorphisms.

In the following sections, we refer to BLS12-377 as $E_{\text{BLS12}}(\mathbb{F}_{q_{\text{BLS12}}})$ with a subgroup of order r_{BLS12} , CP6-782 as $E_{\text{CP6}}(\mathbb{F}_{q_{\text{CP6}}})$ with a subgroup of order r_{CP6} and our curve as $\tilde{E}(\mathbb{F}_{\tilde{q}})$ with a subgroup of order \tilde{r} (cf. Fig. 1).

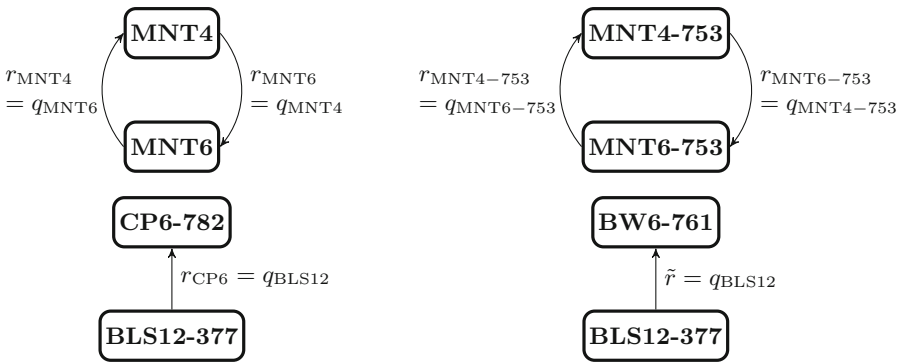


Fig. 1. Examples of pairing-friendly amicable cycles and chains. The arrow signifies that the field arithmetic of the starting curve can be expressed natively *in the exponents* on the second curve.

3 The Proposed Elliptic Curve: BW6-761

The parameters of the two curves proposed in [7] for one-layer proof-composition, BLS12-377 and CP6-782, are given in Table 2. Note that because the Cocks–Pinch method has a ratio $\rho \approx 2$, the CP6-782 curve characteristic q_{CP6} is 782-bit long (832 bits in the Montgomery domain). Since q_{CP6} is already very large, an embedding degree $k = 6$ is sufficient for the security of $\mathbb{F}_{q_{\text{CP6}}}^k$. Moreover, because $D = 339$, E_{CP6} has only a quadratic twist E' and thus $\mathbb{G}_2 \subset E(\mathbb{F}_{q_{\text{CP6}}}^6)[r_{\text{CP6}}]$ is isomorphic to $E'(\mathbb{F}_{q_{\text{CP6}}}^3)[r_{\text{CP6}}]$, and \mathbb{G}_2 elements can be compressed to only $3 \times 832 = 2496$ bits.

Since we are stuck with $\rho \approx 2$, we searched for a Cocks–Pinch curve \tilde{E} with $k = 6$ and the smallest suitable \tilde{q} less or equal to 768 bits. We restricted our search to curves with CM discriminant $D = 3$ to allow optimal \mathbb{G}_2 compression (a sextic twist $\tilde{E}'/\mathbb{F}_{\tilde{q}}$ of \tilde{E} such that \mathbb{G}_2 is isomorphic to $\tilde{E}'(\mathbb{F}_{\tilde{q}})[r]$ of 768 bits)

Table 2. Parameters of BLS12-377 and CP6-782 curves

name	curve type	k	D	r	q	compressed \mathbb{G}_1 in bits	compressed \mathbb{G}_2 in bits
E_{BLS12}	BLS12	12	3	r_{BLS12}	q_{BLS12}	384	768
E_{CP6}	short Weierstrass	6	339	$r_{\text{CP6}} = q_{\text{BLS12}}$	q_{CP6}	832	2496
prime	value x					size in bits	2-adicity of $x - 1$
r_{BLS12}	0x12ab655e9a2ca55660b44d1e5c37b00159aa76fed00000010a11800000000001					253	47
$q_{\text{BLS12}} = r_{\text{CP6}}$	0x1ae3a4617c510eac63b05c06ca1493b1a22d9f300f5138f1ef3622fba094800170b5d44300000008508c0000000001					377	46
q_{CP6}	0x3848c4d2263babf8941fe959283d8f526663bc5d176b746af0266a7223ee72023d07830c728d80f9d78bab3596c8617c579252a3fb77c79c13201ad533049cfe6a399c2f764a12c4024bee135c065f4d26b7545d85c16dfd424adace79b57b942ae9					782	3

Table 3. Parameters of our curve

name	curve type	k	D	r	q	compressed \mathbb{G}_1 in bits	compressed \mathbb{G}_2 in bits
\tilde{E}	short Weierstrass	6	3	$\tilde{r} = q_{\text{BLS12}}$	\tilde{q}	768	768
prime	value x					size in bits	2-adicity of $x - 1$
$\tilde{r} = q_{\text{BLS12}}$	0x1ae3a4617c510eac63b05c06ca1493b1a22d9f300f5138f1ef3622fba094800170b5d44300000008508c0000000001					377	46
\tilde{q}	0x122e824fb83ce0ad187c94004faff3eb926186a81d14688528275ef8087be41707ba638e584e91903cebaff25b423048689c8ed12f9fd9071dcd3dc73ebff2e98a116c25667a8f8160cf8aeef0a437e6913e6870000082f49d00000000008b					761	1

and GLV fast scalar multiplication [18] on \mathbb{G}_1 and \mathbb{G}_2 . Then, following the work of [21], we computed the polynomial form of \tilde{q} , which allowed us to compute the coefficients of an optimal lattice-based final exponentiation as in [17] and perform faster subgroup checks. Finally, the constructed curve has a 2-torsion point allowing fast and secure Elligator 2 hashing-to-point [5]. We also investigate Wahby and Boneh’s work [30] as an alternative hashing method.

The short Weierstrass forms of the curve \tilde{E} and its sextic twist \tilde{E}' are

$$\tilde{E}/\mathbb{F}_{\tilde{q}}: y^2 = x^3 - 1 \quad (4)$$

$$\tilde{E}'/\mathbb{F}_{\tilde{q}}: y^2 = x^3 + 4 \quad (5)$$

and the parameters are given in Table 3.

Given that q_{BLS12} is parameterized by a polynomial $q(u)$ with

$$u = 0x8508c00000000001, \quad (6)$$

we can apply the Brezing–Weng method (cf. Algorithm 3) with $k = 6$, $D = 3$ and $\tilde{r}(x) = (x^6 - 2x^5 + 2x^3 + x + 1)/3$. There are two primitive 6-th roots of

unity in $\mathbb{Q}[x]/\tilde{r}(x)$, and two sets of solutions

$$\begin{cases} \tilde{t}_0(x) = x^5 - 3x^4 + 3x^3 - x + 3 \\ \tilde{y}_0(x) = (x^5 - 3x^4 + 3x^3 - x + 3)/3 \end{cases} \quad \text{or} \quad \begin{cases} \tilde{t}_1(x) = -x^5 + 3x^4 - 3x^3 + x \\ \tilde{y}_1(x) = (x^5 - 3x^4 + 3x^3 - x)/3 \end{cases}$$

Unfortunately neither $(\tilde{t}_0(x) + 3\tilde{y}_0(x))/4$ nor $(\tilde{t}_1(x) + 3\tilde{y}_1(x))/4$ are irreducible polynomials, and we cannot construct a polynomial family of amicable 2-chain elliptic curves. But following [21], with well-chosen lifting cofactors h_t and h_y , we can obtain valid parameters $\tilde{t} = \tilde{r} \times h_t + \tilde{t}_i(u)$ and respectively $\tilde{y} = \tilde{r} \times h_y + \tilde{y}_i(u)$ for $i \in \{0, 1\}$. We found these parameters to be $i = 0$, $h_t = 13$, $h_y = 9$. The polynomial form of the parameters are summarized in Table 4. We propose to name our curve BW6-761 as it is a Brezing–Weng curve of embedding degree 6 over a 761-bit prime field.

Table 4. Polynomial parameters of BW6-761.

BW6-761, $k = 6$, $D = 3$, $\tilde{r} = q_{\text{BLS12}}$
$\tilde{r}(x) = (x^6 - 2x^5 + 2x^3 + x + 1)/3$
$\tilde{t}(x) = x^5 - 3x^4 + 3x^3 - x + 3 + h_t \tilde{r}(x)$
$\tilde{y}(x) = (x^5 - 3x^4 + 3x^3 - x + 3)/3 + h_y \tilde{r}(x)$
$\tilde{q}(x) = (\tilde{t}^2 + 3\tilde{y}^2)/4$
$\tilde{q}_{h_t=13, h_y=9}(x) = (103x^{12} - 379x^{11} + 250x^{10} + 691x^9 - 911x^8 - 79x^7 + 623x^6 - 640x^5 + 274x^4 + 763x^3 + 73x^2 + 254x + 229)/9$

3.1 Optimizations in \mathbb{G}_1

We present some optimizations for widely used operations in cryptography: scalar multiplication, hashing-to-point and subgroup check in \mathbb{G}_1 .

GLV Scalar Multiplication. We have $\tilde{q} \equiv 1 \pmod{3}$ and $\tilde{E}(\mathbb{F}_{\tilde{q}})$ has j -invariant 0. Let ω be an element of order 3 in $\mathbb{F}_{\tilde{q}}$. Then the endomorphism $\phi : \tilde{E} \rightarrow \tilde{E}$ defined by $(x, y) \mapsto (\omega x, y)$ (and $\mathcal{O} \mapsto \mathcal{O}$) acts on a point $P \in \tilde{E}(\mathbb{F}_{\tilde{q}})[\tilde{r}]$ as $\phi(P) = [\lambda]P$ where λ is an integer satisfying $\lambda^2 + \lambda + 1 \equiv 0 \pmod{\tilde{r}}$. Since we expressed \tilde{q} and \tilde{r} as polynomials, we find $\omega(x)$ and $\lambda(x)$ such that

$$\begin{aligned} \omega(x)^2 + \omega(x) + 1 &\equiv 0 \pmod{\tilde{q}(x)} \quad (\text{cube root of unity}) \\ \lambda(x)^2 + \lambda(x) + 1 &\equiv 0 \pmod{\tilde{r}(x)} \end{aligned}$$

where $\lambda_1(x) = x^5 - 3x^4 + 3x^3 - x + 1$, $\lambda_2(x) = -\lambda_1 - 1$, $\omega_1(x) = (103x^{11} - 482x^{10} + 732x^9 + 62x^8 - 1249x^7 + 1041x^6 + 214x^5 - 761x^4 + 576x^3 + 11x^2 - 265x + 66)/21$, $\omega_2(x) = -\omega_1(x) - 1$. Evaluating the polynomials at u (6), we find that for $P(x, y) \in \tilde{E}(\mathbb{F}_{\tilde{q}})$ of order \tilde{r} ,

$$[\lambda_1]P = (\omega_1 x, y), \quad [-\lambda_1 - 1]P = ((-\omega_1 - 1)x, y), \quad \text{where} \tag{7}$$

Algorithm 4: Elligator 2

Input: Θ an octet string to be hashed, $A = 3, B = 3$ coefficients of the curve \tilde{M} and N a constant non-square in $\mathbb{F}_{\tilde{q}}$

Output: A point (x, y) in $\tilde{M}(\mathbb{F}_{\tilde{q}})$

```

1 Define  $g(x) = x(x^2 + Ax + B)$ ;
2  $u = \text{hash2base}(\Theta)$ ;
3  $v = -A/(1 + Nu^2)$ ;
4  $e = \text{Legendre}(g(v), \tilde{q})$ ;
5 if  $u \neq 0$  then
6   |  $x = ev - (1 - e)A/2$ ;
7   |  $y = -e\sqrt{g(x)}$ ;
8 else
9   |  $x = 0$  and  $y = 0$ ;
10 return  $(x, y)$ ;
```

$\lambda_1 = 0x9b3af05dd14f6ec619aaf7d34594aabc5ed1347970dec00452217cc90000008508c00000000001$

$w_1 = 0x531dc16c6ecd27aa846c61024e4cca6c1f31e53bd9603c2d17be416c5e4426ee4a737f73b6f952ab5e57926fa701848e0a235a0a398300c65759fc45183151f2f082d4dcb5e37cb6290012d96f8819c547ba8a4000002f962140000000002a$

Hashing-to-Point. Elligator 2 [5] is an injective map to any elliptic curve of the form $y^2 = x^3 + Ax^2 + Bx$ with $AB(A^2 - 4B) \neq 0$ over any finite field of odd characteristic. Since the point $(1, 0) \in \tilde{E}(\mathbb{F}_{\tilde{q}})$ is of order 2, we can map \tilde{E} to a curve of Montgomery form, precisely $\tilde{M}(\mathbb{F}_{\tilde{q}}) : y^2 = x^3 + 3x^2 + 3x$ (cf. Algorithm 4). In Algorithm 4, $\text{hash2base}(\cdot)$ is a cryptographic hash function to $\mathbb{F}_{\tilde{q}}$, and $\text{Legendre}(a, \tilde{q})$ is the Legendre symbol of an integer a modulo \tilde{q} which is of value 1, $-1, 0$ for when the input is a quadratic residue, non-quadratic-residue, or zero respectively. Elligator 2 is parametrized by a non-square N , for which finding a candidate is an easy computation in general since about half of the elements of $\mathbb{F}_{\tilde{q}}$ are non-squares. For efficiency it is desirable to choose N to be small, or otherwise in a way to speed up multiplications by N . In our case, we can choose $N = -1$ because $\tilde{q} \equiv 3 \pmod{4}$.

Wahby and Boneh introduced in [30] an *indirect* map for the BLS12-381 curve [6] based on the simplified SWU map [8], which works by mapping to an isogenous curve with nonzero j -invariant, then evaluating the isogeny map. We hence check if \tilde{E} has a low-degree rational isogeny. Since 2×11 divides \tilde{y} (CM equation $4\tilde{q} = \tilde{t}^2 + D\tilde{y}^2$), there should be isogenies of degree 2 and 11. Observing that the point $(1, 0)$ is a 2-torsion point, we obtain the rational 2-isogeny $(x, y) \mapsto ((x^2 - x + 3)/(x - 1), y(x^2 - 2x - 2)/(x - 1)^2)$ to the curve $y^2 = x^3 - 15x - 22$ of j -invariant 54000. A 2-torsion point on this curve is $(-2, 0)$

and the dual isogeny to work with Wahby–Boneh hash function is $(x', y') \mapsto ((x'^2 + 2x' - 3)/(x' + 2), y(x'^2 + 4x' + 7)/(x'^2 + 2)^2)$.

Clearing Cofactor. Another important step is clearing the cofactor. Our curve has a large 384-bit long cofactor due to the Cocks–Pinch method. The cofactor has a polynomial form in the seed u like the other parameters, $\tilde{c}(x) = (103x^6 - 173x^5 - 96x^4 + 293x^3 + 21x^2 + 52x + 172)/3$ and $\tilde{c}(x)\tilde{r}(x) = \tilde{q}(x) + 1 - \tilde{t}(x)$. Because the curve has j -invariant 0, it has a fast endomorphism $\phi : (x, y) \mapsto (\omega_1 x, y)$ such that $\phi^2 + \phi + 1$ is the identity map. We apply the same technique as in [17]. First we compute the eigenvalue $\lambda(x)$ of the endomorphism modulo the cofactor. We obtain $\lambda(x) = (-385941x^5 + 1285183x^4 - 1641034x^3 - 121163x^2 + 1392389x - 1692082)/1250420 \pmod{\tilde{c}(x)}$. Then we reduce with LLL the lattice spanned by the rows of the matrix M and obtain a reduced matrix B :

$$M = \begin{bmatrix} \tilde{c}(x) & 0 \\ \lambda(x) \pmod{\tilde{c}(x)} & 1 \end{bmatrix}, \quad B = \frac{1}{103} \begin{bmatrix} 103x^3 - 83x^2 - 40x + 136 & 7x^2 + 89x + 130 \\ -7x^2 - 89x - 130 & 103x^3 - 90x^2 - 129x + 6 \end{bmatrix}.$$

We check that $b_{i0}(x) + \lambda(x)b_{i1}(x) = 0 \pmod{\tilde{c}(x)}$ and $\gcd(b_{i0}(x) + \lambda(x)b_{i1}(x), \tilde{r}(x)) = 1$. Now $x = u$ and for clearing the cofactor, we first precompute uP, u^2P, u^3P which costs three scalar multiplications by u (6), then we compute $R = 103(u^3P) - 83(u^2P) - 40(uP) + 136P + \phi(7(u^2P) + 89(uP) + 130P)$, and R has order \tilde{r} . This formula is compatible with ω_1 to compute ϕ .

Subgroup Check. The subgroup check can benefit from the same technique. We need to check if a point $P \in \tilde{E}(\mathbb{F}_{\tilde{q}})$ has order \tilde{r} , that is, $\tilde{r}P = \mathcal{O}$. Instead of multiplying by \tilde{r} of 377 bits, we can use the endomorphism ϕ as above. This time we need $\lambda_1(x) = x^5 - 3x^4 + 3x^3 - x + 1$ modulo $\tilde{r}(x)$. We reduce the matrix M and obtain a short basis B and check that $(x + 1) + \lambda_1(x)(x^3 - x^2 + 1) = 0 \pmod{\tilde{r}(x)}$.

$$M = \begin{bmatrix} \tilde{r}(x) & 0 \\ -\lambda_1(x) \pmod{\tilde{r}(x)} & 1 \end{bmatrix} \quad B = \begin{bmatrix} x + 1 & x^3 - x^2 + 1 \\ x^3 - x^2 - x & -x - 1 \end{bmatrix}$$

For a faster subgroup check of a point P , one can precompute uP, u^2P, u^3P with three scalar multiplications by u , and check that $uP + P + \phi(u^3P - u^2P + P)$ is the point at infinity.

3.2 Optimizations in \mathbb{G}_2

We present some optimizations for widely used operations on curves: scalar multiplication, hashing-to-point and subgroup check in \mathbb{G}_2 .

Compression of \mathbb{G}_2 Elements. Since the CM discriminant of $\tilde{E}/\mathbb{F}_{\tilde{q}}$ is $D = 3$, the curve has a twist of degree $d = 6$. Additionally, because the embedding degree is $k = 6$, the \tilde{r} -torsion of $\mathbb{G}_2 \subset \tilde{E}[\tilde{r}](\mathbb{F}_{\tilde{q}^6})$ is isomorphic to $\tilde{E}'[\tilde{r}](\mathbb{F}_{\tilde{q}^{k/d}}) = \tilde{E}'[\tilde{r}](\mathbb{F}_{\tilde{q}})$. Thus, elements in \mathbb{G}_2 can be compressed from 4608 bits to 768 bits. We choose the irreducible polynomial $x^6 + 4$ in $\mathbb{F}_{\tilde{q}}[x]$ and construct the M-twist curve $\tilde{E}'/\mathbb{F}_{\tilde{q}}: y^2 = x^3 + 4$. To map a point $Q(x, y) \in \tilde{E}(\mathbb{F}_{\tilde{q}^6})$ to a point on the M-twist curve we use $\xi : (x, y) \mapsto (x/\nu^2, y/\nu^3)$ where $\nu^6 = -4$.

GLV Scalar Multiplication. Since the group \mathbb{G}_2 is isomorphic to the \tilde{r} -torsion in $\tilde{E}'(\mathbb{F}_{\tilde{q}})$ (Eq. (5)), we can apply almost the same GLV decomposition as in equation (7). Given that the order of E' is $\tilde{q} + 1 - (3\tilde{y} + t)/2$, we find that $[\lambda_1]P = (\omega_2x, y)$ and $[\lambda_2]P = (\omega_1x, y)$ for any $P(x, y) \in E'[\tilde{r}]$.

Hashing-to-Point. The curve $\tilde{E}'(\mathbb{F}_{\tilde{q}}) : y^2 = x^3 + 4$ doesn't have a point of order 2 so Elligator 2 doesn't apply. Furthermore, we didn't find a low-degree rational isogeny and thus Wahby–Boneh method is not efficiently applicable. However, we can apply the more generic Shallue–Woestijn algorithm [28] that works for any elliptic curve over a finite field \mathbb{F}_q of odd characteristic with $\#\mathbb{F}_q > 5$. It runs deterministically in time $O(\log^4 q)$ or in time $O(\log^3 q)$ if $q \equiv 3 \pmod 4$ (a square root costs an exponentiation). Fouque and Tibouchi [15] adapted this algorithm to BN curves ($j = 0$) assuming that $q \equiv 7 \pmod{12}$ and that the point $(1, y) \neq (1, 0)$ lies on the curve. Noting that $\tilde{E}' : y^2 = f(x) = x^3 + 4$ satisfies these assumptions, we propose using Fouque–Tibouchi map for BW6-761:

$$g : \mathbb{F}_{\tilde{q}}^* \rightarrow \tilde{E}'(\mathbb{F}_{\tilde{q}})$$

$$z \mapsto \left(x_i(z), \text{Legendre}(z, \tilde{q}) \times \sqrt{f(x_i(z))} \right)_{i \in \{1, 2, 3\}}$$

where

$$x_1(z) = \frac{-1 + \sqrt{-3}}{2} - \frac{\sqrt{-3} \cdot z^2}{5 + z^2} ; \quad x_2(z) = \frac{-1 - \sqrt{-3}}{2} + \frac{\sqrt{-3} \cdot z^2}{5 + z^2} ; \quad x_3(z) = 1 - \frac{(5 + z^2)^2}{3z^2}$$

Because of the Skalba identity $y^2 = f(x_1) \cdot f(x_2) \cdot f(x_3)$, at least one of the x_i above is an abscissa of a point on \tilde{E}' —we choose the smallest index i such that $f(x_i)$ is a square in $\mathbb{F}_{\tilde{q}}$.

Clearing Cofactor. Given a point $P' \in \tilde{E}'(\mathbb{F}_{\tilde{q}})$, we precompute uP', u^2P' and u^3P' with three scalar multiplications, then the point R' has order \tilde{r} , with

$$R' = (103u^3P' - 83u^2P' - 143uP' + 27P') + \phi(7u^2P' - 117uP' - 109P') .$$

Subgroup Check. The formula compatible with ω_1 for ϕ is

$$-uP' - P' + \phi(u^3P' - u^2P' - uP') = 0 \iff \tilde{r}P' = \mathcal{O} .$$

3.3 Pairing Computation

The verification equation of proof composition (3) requires three pairing computations. When an even-degree twist is available, the denominators $v_{[2]S}(P)$, $v_{S+Q}(P)$ in Algorithm 1 are in a proper subgroup of \mathbb{F}_{q^k} and can be removed as they resolve to one after the final exponentiation. This is the case for BLS12, CP6-782 and BW6-761. We estimate, in terms of multiplications in the base fields $\mathbb{F}_{\tilde{q}}$, an optimal ate pairing on BW6-761. We follow the estimate in [21]. We model the cost of arithmetic in a degree 6, resp. degree 12 extension in the usual way, where multiplications and squarings in quadratic and cubic extensions are

obtained recursively with Karatsuba and Chung–Hasan formulas, summarized in Table 5. We denote by \mathbf{m}_k , \mathbf{s}_k , \mathbf{i}_k and \mathbf{f}_k the costs of multiplication, squaring, inversion, and q -th power Frobenius in an extension \mathbb{F}_{q^k} , and by $\mathbf{m} = \mathbf{m}_1$ the multiplication in a base field \mathbb{F}_q . We neglect additions and multiplications by small constants. Below, we compute the formulae of an ate pairing and an optimized final exponentiation for BW6-761. We also provide timings of these formulae based on a Rust implementation and compare them to CP6-782 timings (Table 6).

Table 5. Cost from [21, Table 6] of \mathbf{m}_k , \mathbf{s}_k and \mathbf{i}_k for finite field extensions involved.

k	1	2	3	6	12
\mathbf{m}_k	\mathbf{m}	$3\mathbf{m}$	$6\mathbf{m}$	$18\mathbf{m}$	$54\mathbf{m}$
\mathbf{s}_k	\mathbf{m}	$2\mathbf{m}$	$5\mathbf{m}$	$12\mathbf{m}$	$36\mathbf{m}$
\mathbf{f}_k	0	0	$2\mathbf{m}$	$4\mathbf{m}$	$10\mathbf{m}$
$\mathbf{s}_k^{\text{cyclo}}$				$6\mathbf{m}$	$18\mathbf{m}$
$\mathbf{i}_k - \mathbf{i}_1$	0	$4\mathbf{m}$	$12\mathbf{m}$	$34\mathbf{m}$	$94\mathbf{m}$
\mathbf{i}_k , with $\mathbf{i}_1 = 25\mathbf{m}$	$25\mathbf{m}$	$29\mathbf{m}$	$37\mathbf{m}$	$59\mathbf{m}$	$119\mathbf{m}$

Table 6. Miller loop cost in Weierstrass model from [1, 13], homogeneous coordinates.

k	D	curve	DOUBLELINE and ADDLINE	UPDATE1 and UPDATE2	ref
$6 \mid k$	-3	$y^2 = x^3 + b$ sextic twist	$2\mathbf{m}_{k/6} + 7\mathbf{s}_{k/6} + (k/3)\mathbf{m}$ $10\mathbf{m}_{k/6} + 2\mathbf{s}_{k/6} + (k/3)\mathbf{m}$	$\mathbf{s}_k + 13\mathbf{m}_{k/6}$ $13\mathbf{m}_{k/6}$	[13, §5]
$6 \mid k$	-3	$y^2 = x^3 + b$ sextic twist	$3\mathbf{m}_{k/6} + 6\mathbf{s}_{k/6} + (k/3)\mathbf{m}$ $11\mathbf{m}_{k/6} + 2\mathbf{s}_{k/6} + (k/3)\mathbf{m}$	$\mathbf{s}_k + 13\mathbf{m}_{k/6}$ $13\mathbf{m}_{k/6}$	[1, §4,6]

Miller Loop for BW6-761. For BW6-761, the Tate pairing has Miller loop length $\tilde{r}(x) = (x^6 - 2x^5 + 2x^3 + x + 1)/3$, and the ate pairing has Miller loop length $\tilde{t}(x) - 1 = (13x^6 - 23x^5 - 9x^4 + 35x^3 + 10x + 19)/3$, hence the ate pairing will be slightly slower. Recall that thanks to a degree 6 twist, the two points $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$ have coordinates in $\mathbb{F}_{\tilde{q}}$, hence swapping the two points for the ate pairing does not slow down the Miller loop in itself. We now apply Vercauteren’s method [29] to obtain an optimal ate pairing of minimal Miller loop. Define the lattice spanned by the rows of the matrix M , and reduce it with LLL. One obtains the short basis B made of linear combinations of M :

$$M = \begin{bmatrix} \tilde{r}(x) & 0 \\ -\tilde{q}(x) \bmod \tilde{r}(x) & 1 \end{bmatrix} \quad B = \begin{bmatrix} x + 1 & x^3 - x^2 - x \\ x^3 - x^2 + 1 & -x - 1 \end{bmatrix}.$$

Algorithm 5: Miller Loop for BW6-761

```

1  $m \leftarrow 1; S \leftarrow Q;$ 
2 for  $b$  from the second most significant bit of  $u$  to the least do
3    $\ell \leftarrow \ell_{S,S}(P); S \leftarrow [2]S;$  DOUBLELINE
4    $m \leftarrow m^2 \cdot \ell;$  UPDATE1
5   if  $b = 1$  then
6      $\ell \leftarrow \ell_{S,Q}(P); S \leftarrow S + Q;$  ADDLINE
7      $m \leftarrow m \cdot \ell;$  UPDATE2
8  $Q_u \leftarrow \text{AffineCoordinates}(S);$  Homogeneous ( $\mathcal{H}$ ):  $\mathbf{i} + 2\mathbf{m}$ ; Jacobian
    $(\mathcal{J}): \mathbf{i} + \mathbf{s} + 3\mathbf{m}$ 
9  $m_{-u} \leftarrow 1/m_u; S \leftarrow Q_u; m_u \leftarrow m;$   $\mathbf{i}_6$ 
10  $\ell \leftarrow \ell_{Q_u,Q}(P); Q_{u+1} \leftarrow Q_u + Q;$  ADDLINE
11  $m_{u+1} \leftarrow m_u \cdot \ell;$  UPDATE2
12 for  $b$  from the second most significant bit of  $(u^2 - u - 1)$  to the least do
13    $\ell \leftarrow \ell_{S,S}(P); S \leftarrow [2]S;$  DOUBLELINE
14    $m \leftarrow m^2 \cdot \ell;$  UPDATE1
15   if  $b = 1$  then
16      $\ell \leftarrow \ell_{S,Q_u}(P); S \leftarrow S + Q_u;$  ADDLINE
17      $m \leftarrow m \cdot m_u \cdot \ell;$   $\mathbf{m}_6 + \text{UPDATE2}$ 
18   else if  $b = -1$  then
19      $\ell \leftarrow \ell_{S,-Q_u}(P); S \leftarrow S - Q_u;$  ADDLINE
20      $m \leftarrow m \cdot m_{-u} \cdot \ell;$   $\mathbf{m}_6 + \text{UPDATE2}$ 
21 return  $m_{u+1} \cdot m^{\tilde{q}};$   $\mathbf{f}_6 + \mathbf{m}_6$ 

```

One can check that $(x + 1) + (x^3 - x^2 - x)\tilde{q}(x) \equiv 0 \pmod{\tilde{r}(x)}$. The optimal ate pairing on our curve is

$$\text{ate}_{\text{opt}}(P, Q) = (f_{u+1,Q}(P) f_{u^3-u^2-u,Q}^{\tilde{q}}(P) \ell_{[u^3-u^2-u]\pi(Q), [u+1]Q}(P))^{(\tilde{q}^6-1)/\tilde{r}}$$

and since $(u+1) + \tilde{q}(u^3 - u^2 - u) = 0 \pmod{\tilde{r}}$, we have $[u+1]Q + \pi_{\tilde{q}}([u^3 - u^2 - u]Q) = \mathcal{O}$. The line $\ell_{[u^3-u^2-u]\pi(Q), [u+1]Q}$ is vertical and can be removed. Finally,

$$\text{ate}_{\text{opt}}(P, Q) = (f_{u+1,Q}(P) f_{u^3-u^2-u,Q}^{\tilde{q}}(P))^{(\tilde{q}^6-1)/\tilde{r}}. \quad (8)$$

Now we share the computation of $f_{u,Q}(P)$ to optimize further the Miller loop, noting that $f_{u+1,Q}(P)$ equals $f_{u,Q}(P) \cdot \ell_{[u]Q,Q}(P)$. We can re-use $f_{u,Q}(P)$ to compute the second part $f_{u(u^2-u-1),Q}$ since $f_{uv,Q} = f_{u,Q}^v f_{v,[u]Q}$. This results in Algorithm 5. We write $u^2 - u - 1$ in 2-non-adjacent-form (2-NAF) to minimize the addition steps in the Miller loop, and replace Q by $-Q$ in the algorithm when the bit b_i is -1 . The scalar $u^2 - u - 1$ is 127-bit long and has $\text{HW}_{2\text{-NAF}} = 19$.

$$\begin{aligned} \text{Cost}_{\text{MILLERLOOP}} &= (\text{nbits}(u) - 1)\text{Cost}_{\text{DOUBLELINE}} + (\text{nbits}(u) - 2)\text{Cost}_{\text{UPDATE1}} \\ &\quad + (\text{HW}(u) - 1)(\text{Cost}_{\text{ADDLINE}} + \text{Cost}_{\text{UPDATE2}}) \\ &\quad + (\text{nbits}(u^2 - u - 1) - 1)(\text{Cost}_{\text{DOUBLELINE}} + \text{Cost}_{\text{UPDATE1}}) \\ &\quad + (\text{HW}_{2\text{-NAF}}(u^2 - u - 1) - 1)(\text{Cost}_{\text{ADDLINE}} + \text{Cost}_{\text{UPDATE2}} + \mathbf{m}_6) \\ &\quad + \mathbf{i} + 2\mathbf{m} + \mathbf{i}_6 + \mathbf{f}_6 + \mathbf{m}_6 \end{aligned}$$

We plug in the values of our curve and obtain $(64-1)(3\mathbf{m}+6\mathbf{s}+2\mathbf{m})+(64-2)(\mathbf{s}_6+13\mathbf{m})+(7-1)(11\mathbf{m}+2\mathbf{s}+2\mathbf{m}+13\mathbf{m})+(127-1)(3\mathbf{m}+6\mathbf{s}+2\mathbf{m}+13\mathbf{m}+\mathbf{s}_6)+(19-1)(11\mathbf{m}+2\mathbf{s}+2\mathbf{m}+13\mathbf{m}+\mathbf{m}_6)+\mathbf{i}+2\mathbf{m}+\mathbf{i}_6+13\mathbf{m}+\mathbf{f}_6+\mathbf{m}_6 = 7861\mathbf{m}+2\mathbf{i} \approx 7911\mathbf{m}$ with $\mathbf{m} = \mathbf{s}$ and $\mathbf{i} \approx 25\mathbf{m}$. We note that since $\tilde{q}(x) \equiv \tilde{t}(x) - 1 \equiv \lambda(x) + 1 \pmod{\tilde{r}(x)}$, the formulas are similar as for the subgroup check in \mathbb{G}_1 in Sect. 3.1, where we found that $(x + 1) + \lambda(x)(x^3 - x^2 + 1) = 0 \pmod{\tilde{r}(x)}$.

Final Exponentiation for BW6-761. Given that \tilde{q} has a polynomial form, we can compute the coefficients of a fast final exponentiation as in [17]. The easy part is raising to $(\tilde{q}^3 - 1)(\tilde{q} + 1)$ and costs $99\mathbf{m}$. For the hard part $\sigma = (\tilde{q}^2 - \tilde{q} + 1)/\tilde{r}$, we raise to a multiple $\sigma'(u)$ of σ with $\tilde{r} \nmid \sigma$. Following [17], we find that $\sigma'(x) = R_0(x) + \tilde{q} \times R_1(x)$ with

$$R_0(x) = -103x^7 + 70x^6 + 269x^5 - 197x^4 - 314x^3 - 73x^2 - 263x - 220$$

$$R_1(x) = 103x^9 - 276x^8 + 77x^7 + 492x^6 - 445x^5 - 65x^4 + 452x^3 - 181x^2 + 34x + 229$$

and a polynomial cofactor $3(x^3 - x^2 + 1)$ to $\sigma(x)$. The exponentiation is dominated by exponentiations to u, u^2, \dots, u^9 . Raising to u costs $\exp_u = 4(\text{nbits}(u) - 1)\mathbf{m} + (6\text{HW}(u) - 3)\mathbf{m} + (\text{HW}(u) - 1)\mathbf{m}_6 + 3(\text{HW}(u) - 1)\mathbf{s} + \mathbf{i} = 4 \cdot 63\mathbf{m} + 39\mathbf{m} + 6\mathbf{m}_6 + 18\mathbf{s} + \mathbf{i} = 417 + \mathbf{i} = 442\mathbf{m}$; and nine such u -powers cost $3978\mathbf{m}$. Eight Frobenius powers $\mathbf{f}_6 = 4\mathbf{m}$ occur, as do some exponentiations to the small coefficients of R_0, R_1 . They do not seem suited for the short addition chain so a multi-exponentiation in 2-NAF technique can reduce the cost to $9\mathbf{s}_6^{\text{cyclo}} + 51\mathbf{m}_6 = 972\mathbf{m}$. The total count is $(99 + 3978 + 32 + 972)\mathbf{m} = 5081\mathbf{m}$.

Table 7. Pairing cost estimation in terms of base field multiplications \mathbf{m}_b , where b is the bitsize in Montgomery domain on a 64-bit platform.

Curve	Prime	Pairing	Miller loop	Exponentiation	Total
BLS12-377	377-bit q_{BLS12}	ate	$6705\mathbf{m}_{384}$	$7063\mathbf{m}_{384}$	$13768\mathbf{m}_{384}$
CP6-782	782-bit q_{CP6}	Tate	$21510\mathbf{m}_{832}$	$10521\mathbf{m}_{832}$	$32031\mathbf{m}_{832}$
		ate	$47298\mathbf{m}_{832}$	$10521\mathbf{m}_{832}$	$57819\mathbf{m}_{832}$
		opt. ate	$21074\mathbf{m}_{832}$	$10521\mathbf{m}_{832}$	$31595\mathbf{m}_{832}$
BW6-761	761-bit \tilde{q}	opt. ate	$7911\mathbf{m}_{768}$	$5081\mathbf{m}_{768}$	$12992\mathbf{m}_{768}$

Finally, according to Table 7, BW6-761 is much faster than CP6-782. We obtain a pairing whose cost in terms of base field multiplications is two times cheaper compared to the Tate pairing on CP6-782 and four times cheaper than the ate pairing available in Zexe Rust implementation [24], and moreover the multiplications take place in a smaller field by one 64-bit machine word. Particularly, it is at least five times cheaper to compute on BW6-761 the product of pairings needed to verify a Groth proof (Eq. (3)). The verification would be even cheaper if we included the GLV-based multi-scalar multiplication in \mathbb{G}_1 needed

for vk_x and subgroup checks in \mathbb{G}_1 and \mathbb{G}_2 needed for the proof and the verification keys. Finally, We note that proof generation is also faster on the new curve but we chose to base the comparison on the verification equation for two reasons: The cost of proof generation depends on the NP-statement being proven while the verification cost is constant for a given curve, and in blockchain applications, only the verification is performed on the blockchain, costing execution fees.

Implementation. We provide a Sagemath script and a Magma script to check the formulas and algorithms of this section. The source code is available under MIT license at <https://gitlab.inria.fr/zk-curves/bw6-761>. We also provide, based on `libzexe` [24], a Rust implementation of the curve arithmetic for \tilde{E} and \tilde{E}' , along with the optimal ate pairing described in Algorithm 5 and the fast final exponentiation in the full version of the paper. The source code was merged into `libzexe` [24] in the pull request 210 and is available under dual MIT/Apache public license at: <https://github.com/scipr-lab/zexe/pull/210>.

For benchmarking, we choose to compare the timings of a pairing computation, and an evaluation of Eq. (3) whose costs are dominated by 3 Miller loops and 1 final exponentiation (cf. Table 8). It is of note that the ZEXE CP6-782 code implements an ate pairing with affine coordinates as those should lead, in the case of that curve, to a faster Miller loop than with projective coordinates, as suggested in [25]. The Rust implementation was tested on a 2.2 GHz Intel Core i7 x86-64 processor with 16 Go 2400 MHz DDR4 memory running macOS Mojave 10.14.6 and compiled with Cargo 1.43.0.

Table 8. Rust implementation timings of ate pairing computation on CP6-782 and optimal ate pairing on BW6-761.

Curve	Pairing	Miller loop	Exponentiation	Total	Eq. 3
CP6-782	ate (affine)	76.1 ms	8.1 ms	84.2 ms	309.4 ms
BW6-761	opt. ate	2.5 ms	3 ms	5.5 ms	10.5 ms

We conclude that using this work, a pairing computation is 15.3 times faster and the verification of a Groth16 proof is 29.5 times faster, compared to the CP6-782 ZEXE implementation. This huge gap between theory and practice is mainly due to the choice of affine coordinates in the Miller loop without implementing optimized inversion in \mathbb{F}_{q^3} as advised in [25]. Finally, one should also include the cost to check that the proof and the verification key elements are in the right subgroups \mathbb{G}_1 and \mathbb{G}_2 , which should also be faster on BW6-761 as described in Sect. 3.1, Sect. 3.2.

4 Security Estimate of the Curves

We estimate the security in \mathbb{G}_T against a discrete logarithm computation for BLS12-377, CP6-782, and BW6-761. BLS12-377 has a security of about 2^{125} in

$\mathbb{F}_{q_{\text{BLS}}^{12}}$, CP6-782 has security about 2^{138} in $\mathbb{F}_{q_{\text{CP6}}^6}$ and BW6-761 has a security of about 2^{126} in $\mathbb{F}_{\tilde{q}^6}$, taking the Special Tower NFS algorithm into account and the model of estimation in [22]. For CP6-782, the characteristic q_{CP6} does not have a special form and we consider the TNFS algorithm with the same set of parameters as for a MNT curve of embedding degree 6. The security on the curve BLS12-377 (\mathbb{G}_1 and \mathbb{G}_2) is 126 bits because r_{BLS12} is 253-bit long. The security in \mathbb{G}_1 and \mathbb{G}_2 for CP6-782 and BW6-761 is 188 bits as $r_{\text{CP6}} = \tilde{r}$ is 377-bit long.

The curve parameters are given in Tables 1 and 2 for BLS12-377 and CP6-782, and Tables 4, 3 for BW6-761. In [22, Table 5], the BLS12-381 curve has a security in \mathbb{G}_T estimated about 2^{126} . We obtain a very similar result for BLS12-377: 2^{125} , indeed the curves are almost the same size. Very luckily, we also obtained a security of 2^{126} in $\mathbb{F}_{\tilde{q}^6}$. The full version of the paper contains the numerical data.

4.1 A Note on Cheon’s Attack

Cheon [11] showed that given G , $[\tau]G$ and $[\tau^T]G$, with G a point in a subgroup \mathbb{G} of order r with $T|r-1$, it is possible to recover τ in $2([\sqrt{(r-1)/T}] + [\sqrt{T}]) \times (\text{Exp}_{\mathbb{G}}(r) + \log r \times \text{Comp}_{\mathbb{G}})$ where $\text{Exp}_{\mathbb{G}}(r)$ stands for the cost of one exponentiation in \mathbb{G} by a positive integer less than r and $\text{Comp}_{\mathbb{G}}$ for the cost to determine if two elements are equal in \mathbb{G} . According to [11, Theorem 2], if $T \leq r^{1/3}$, then the complexity of the attack is about $O(\sqrt{r/T})$ exponentiation by using $O(\sqrt{r/T})$ storage.

In zkSNARK settings such as in [20], the preprocessing phase includes elements $[\tau^i]_{i=0}^T G_1 \in \mathbb{G}_1$ and $[\tau^i]_{i=0}^T G_2 \in \mathbb{G}_2$, where $T \in \mathbb{N}^*$ is the size of the arithmetic circuit related to the NP-statement being proven, and τ is a secret trapdoor. The property $T | r - 1$ also holds since we need r to be highly *2-adic* (we have $2^{47} | r - 1$ and $2^{46} | \tilde{r} - 1$). So, given these auxiliary inputs, an attacker could recover the secret using Cheon’s algorithm in time $O(\sqrt{r/T})$, hence breaking the zkSNARK soundness. We estimate the security of the curves BLS12-377 and BW6-761 with the applications we mentioned, precisely the Nightfall fungible tokens transfer circuit ($T_{\mathbb{G}_1} = 2^{22} - 1, T_{\mathbb{G}_2} = 2^{21}$) and the Filecoin circuit ($T_{\mathbb{G}_1} = 2^{28} - 1, T_{\mathbb{G}_2} = 2^{27}$), which is the biggest circuit of public interest in the blockchain community.

We recall that our curve is designed for proof composition and that the verifier circuit of a Groth’s proof can be expressed in $\tilde{T} = 40000$ constraints. Hence, the complexity of Cheon’s attack on BW6-761, in the case of composing Groth’s proofs, would be $O(\sqrt{\tilde{r}/\tilde{T}}) \approx 2^{181}$. However, for completeness, we state that the curve still has at least 126 bits of security as previously stated, for circuits of size up to 2^{124} which is, to the authors’ knowledge, “large enough” for all the published applications.

For the BLS12-377 curve, because the subgroup order is 253-bit long, the estimated security for Nightfall setup is about 116 bits in \mathbb{G}_1 and for Filecoin, about 113 bits in \mathbb{G}_1 . While this curve has a standard security of 128 bits under

generic attacks (without auxiliary inputs), one should use it with small SNARK circuits or look for alternative inner curves, which we set as a future work.

5 Conclusion

In this work, we construct a new pairing-friendly elliptic curve on top of Zexe's inner curve which is suitable for one layer proof composition. We discuss its security and the optimizations it allows. We conclude that it is at least five times faster for verifying Groth's proof compared to the previous state-of-the-art, and validate our results with an open-source Rust implementation. We mentioned several projects in the blockchain community that need one layer proof composition and therefore can benefit from this work. Applications such as Zexe, EY Nightfall, Celo or Filecoin can consequently reduce their operation cost.

References

1. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster explicit formulas for computing pairings over ordinary curves. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 48–68. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_5
2. Arène, C., Lange, T., Naehrig, M., Ritzenthaler, C.: Faster computation of the tate pairing. *J. Number Theory* **131**(5, Elliptic Curve Cryptography), 842–857 (2011). <https://doi.org/10.1016/j.jnt.2010.05.013>, <http://cryptojedi.org/papers/#edpair>
3. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_6
4. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 276–294. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_16
5. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: elliptic-curve points indistinguishable from uniform random strings. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 967–980. ACM Press, November 2013. <https://doi.org/10.1145/2508859.2516734>
6. Bowe, S.: BLS12-381: New zk-SNARK elliptic curve construction (2017). <https://electriccoin.co/blog/new-snark-curve/>
7. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: Zexe: enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy (SP), Los Alamitos, CA, USA, pp. 1059–1076. IEEE Computer Society, May 2020. <https://www.computer.org/csdl/proceedings-article/sp/2020/349700b059/1i0rIqoBYD6>
8. Brier, E., Coron, J.-S., Icart, T., Madore, D., Randriam, H., Tibouchi, M.: Efficient indifferentiable hashing into ordinary elliptic curves. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 237–254. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_13
9. Celo: BLS-ZEXE: BLS signatures verification inside a SNARK proof (2019). <https://github.com/celo-org/bls-zexe>

10. Chatterjee, S., Sarkar, P., Barua, R.: Efficient computation of Tate pairing in projective coordinate over general characteristic fields. In: Park, C., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 168–181. Springer, Heidelberg (2005). https://doi.org/10.1007/11496618_13
11. Cheon, J.H.: Discrete logarithm problems with auxiliary inputs. *J. Cryptol.* **23**(3), 457–476 (2009). <https://doi.org/10.1007/s00145-009-9047-0>
12. Chiesa, A., Chua, L., Weidner, M.: On cycles of pairing-friendly elliptic curves. *SIAM J. Appl. Algebra Geo.* **3**(2), 175–192 (2019). <https://doi.org/10.1137/18M1173708>
13. Costello, C., Lange, T., Naehrig, M.: Faster pairing computations on curves with high-degree twists. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 224–242. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_14
14. EY-Blockchain: Nightfall: an open source suite of tools designed to enable private token transactions over the public Ethereum blockchain (2019). <https://github.com/EYBlockchain/nightfall>
15. Fouque, P.-A., Tibouchi, M.: Indifferentiable hashing to Barreto–Naehrig curves. In: Hevia, A., Neven, G. (eds.) LATINCRYPT 2012. LNCS, vol. 7533, pp. 1–17. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33481-8_1
16. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. *J. Cryptol.* **23**(2), 224–280 (2010). <https://doi.org/10.1007/s00145-009-9048-z>
17. Fuentes-Castañeda, L., Knapp, E., Rodríguez-Henríquez, F.: Faster Hashing to \mathbb{G}_2 . In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 412–430. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28496-0_25
18. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_11
19. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18**(1), 186–208 (1989). <https://doi.org/10.1137/0218012>
20. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
21. Guillevic, A., Masson, S., Thomé, E.: Cocks–Pinch curves of embedding degrees five to eight and optimal ate pairing computation. *Des. Codes Crypt.* **88**(6), 1047–1081 (2020). <https://doi.org/10.1007/s10623-020-00727-w>
22. Guillevic, A., Singh, S.: On the alpha value of polynomials in the tower number field sieve algorithm. *Cryptology ePrint Archive, Report 2019/885* (2019). <https://eprint.iacr.org/2019/885>
23. Karabina, K.: Squaring in cyclotomic subgroups. *Math. Comput.* **82**(281), 555–579 (2013). <https://doi.org/10.1090/S0025-5718-2012-02625-1>
24. scipr lab: ZEXE rust implementation (2018). <https://github.com/scipr-lab/zexe>
25. Lauter, K., Montgomery, P.L., Naehrig, M.: An analysis of affine coordinates for pairing computation. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 1–20. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17455-1_1
26. Meckler, I., Shapiro, E.: Coda: decentralized cryptocurrency at scale. O(1) Labs whitepaper (2018). <https://cdn.codaprotocol.com/v2/static/coda-whitepaper-05-10-2018-0.pdf>

27. ProtocolLabs: Filecoin: a decentralized storage network (2017). <https://filecoin.io/filecoin.pdf>
28. Shallue, A., van de Woestijne, C.E.: Construction of rational points on elliptic curves over finite fields. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 510–524. Springer, Heidelberg (2006). https://doi.org/10.1007/11792086_36
29. Vercauteren, F.: Optimal pairings. *IEEE Trans. Inf. Theory* **56**(1), 455–461 (2010). <https://doi.org/10.1109/TIT.2009.2034881>
30. Wahby, R.S., Boneh, D.: Fast and simple constant-time hashing to the BLS12-381 elliptic curve. *IACR TCHES* **2019**(4), 154–179 (2019). <https://doi.org/10.13154/tches.v2019.i4.154-179>



Curves with Fast Computations in the First Pairing Group

Rémi Clarisse^{1,2(✉)}, Sylvain Duquesne², and Olivier Sanders¹

¹ Orange Labs, Applied Crypto Group, Cesson-Sévigné, France
`remi.clarisse@univ-rennes1.fr`

² Univ. Rennes, CNRS, IRMAR - UMR 6625, 35000 Rennes, France

Abstract. Pairings are a powerful tool to build advanced cryptographic schemes. The most efficient way to instantiate a pairing scheme is through Pairing-Friendly Elliptic Curves.

Because a randomly picked elliptic curve will not support an efficient pairing (the embedding degree will usually be too large to make any computation practical), a pairing-friendly curve has to be carefully constructed. This has led to famous curves, *e.g.* Barreto-Naehrig curves.

However, the computation of the Discrete Logarithm Problem on the finite-field side has received much interest and its complexity has recently decreased. Hence the need to propose new curves has emerged.

In this work, we give one new curve that is specifically tailored to be fast over the first pairing-group, which is well suited for several cryptographic schemes, such as group signatures, and their variants, or accumulators.

1 Introduction

Pairings and cryptography have a long common history. Initially used as a way to shift the discrete logarithm problem from elliptic curves to finite fields [35], it has first been used for constructive purpose by Joux [30] in 2000. Following this seminal result, pairings have been massively used in cryptography. This is due in large part to the nice features of this mathematical tool but also to its apparent simplicity. Indeed, the features of pairings can easily be abstracted so as to be used even by non-specialists. Actually, almost all pairing-based cryptographic papers (*e.g.* [6, 10, 37]) consider so-called “bilinear groups” as some kind of black box given by a set of three groups \mathbb{G}_1 , \mathbb{G}_2 and \mathbb{G}_T (usually of prime order ℓ) along with an efficiently-computable non-degenerate bilinear map between them

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T.$$

This way, cryptographers can design their protocols without being bothered by the technicalities of the concrete implementations of pairings. The only subtlety considered in cryptographic papers is the existence of efficiently computable morphisms between \mathbb{G}_1 and \mathbb{G}_2 , which has led to so called *type- i* pairings, for $i \in \{1, 2, 3\}$, according to the classification by Galbraith *et al.* [23]. However,

type-3 pairings are now preponderant in cryptography because they are the most efficient ones [23] and because they are compatible with cryptographic assumptions, such as Decisional Diffie-Hellman in both \mathbb{G}_1 and \mathbb{G}_2 , that do not hold with the other types. Actually, some recent results [1, 16] cast some doubts on the real interest of type-1 and type-2 pairings for cryptography. For all these reasons, we only consider type-3 pairings in this paper.

In all cases, at some point, it becomes necessary to instantiate these bilinear groups. To date, the only secure instantiations are (to our knowledge) based on elliptic curves as the constructions based on lattices [19, 25] have been proved insecure [17, 18]. More specifically, on one hand, \mathbb{G}_1 and \mathbb{G}_2 are usually defined as cyclic groups of prime order ℓ of some elliptic curve E over a finite field \mathbb{F}_q . On the other hand, \mathbb{G}_T is the group of ℓ -th roots of unity in \mathbb{F}_{q^k} , where k is the order of q in $\mathbb{Z}/\ell\mathbb{Z}$, called the embedding degree of q .

It is thus important to understand that, despite being considered as similar objects by the cryptographic abstraction of bilinear groups, the groups \mathbb{G}_1 , \mathbb{G}_2 and \mathbb{G}_T are extremely different in practice. The main difficulty when it comes to instantiate these groups is to carefully select the different parameters (essentially ℓ , q and k) in order to ensure that security holds in each group while retaining the best efficiency. Here, by “security” we mean the hardness of the Discrete Logarithm Problem (DLP) although cryptographic schemes usually rely on easier problems.

Actually, for a long time, the problem of selecting these parameters was thought to be rather easy. It was indeed thought that the hardness of the DLP problem in \mathbb{F}_{q^k} only depended on the bitlength of this field (namely $k \log_2(q)$) and not on k and q themselves. Using this assumption, it was quite simple to derive concrete bounds on ℓ , q and k to achieve some specific security level λ . For example, in the standard case $\lambda = 128$, a simple look at [42, Table 4.6] reveals that we must have $\log_2(\ell) \geq 256$ (and so $\log_2(q) \geq 256$ because of the Hasse’s bound) and $k \log_2(q) \geq 3072$. In this case, parameters $q \sim \ell \sim 2^{256}$ and $k = 12$ are optimal. Moreover the choice of an even k leads to very efficient implementations of pairings and of the arithmetic in \mathbb{F}_{q^k} . This largely explains the success of the so-called Barreto-Naehrig curves (BN) [8] (that perfectly match these parameters) which have become *de facto* the standard pairing curves in the literature.

Unfortunately, two recent papers [32, 33] have shown that the assumption regarding \mathbb{F}_{q^k} was wrong. We will discuss the details later but intuitively these results imply that the bitlength of the elements of \mathbb{F}_{q^k} is no longer the good metric to estimate security in this group as it now depends on the shape of both integers q and k . We now face a somewhat chaotic situation where a 3000-bit finite field \mathbb{F}_{q^k} may offer the same security level as a 5000-bit one provided that k and q have some specific properties. And \mathbb{G}_T is not the only group concerned by these considerations as the parameter q has a direct impact on \mathbb{G}_1 and \mathbb{G}_2 . Concretely, it is now sometimes necessary to significantly increase the parameter q (that becomes much larger than the advised minimal bound 2^{256}) to remain compatible with some values of k that enables efficient pairing computations. This

is illustrated by the BLS12 curves promoted by Barbulescu and Duquesne’s [4] for the standard 128-bits level of security. These curves lead to a 450-bit prime integer q , *i.e.* 75% higher than old BN curves.

These examples are representative of the current strategy to select pairing parameters [4]. The goal is indeed to find a nice compromise between the complexity of the different groups/operations. More specifically, it aims at providing parameters that would not significantly penalize one particular operation. This strategy thus makes the implicit assumption that cryptographic protocols present some kind of symmetry, between the groups \mathbb{G}_1 , \mathbb{G}_2 and \mathbb{G}_T , but also between the entities that will perform the operations in these groups. This may be true in some specific scenarios but there are many others were this assumption is false.

As an example, let us consider the case of Enhanced Privacy ID (EPID) scheme introduced by Brickell and Li [14] and now massively used to secure Intel SGX enclave [2]. EPID is a variant of Direct Anonymous Attestation (DAA) [12] with enhanced revocation capabilities, meaning that it is possible to revoke a secret key sk by simply adding a signature generated using sk to a revocation list. The next signatures will then have to contain a proof that they were issued using a different secret key. This is a nice feature but it implies to perform a high number of exponentiations in \mathbb{G}_1 [13], linear in the number n of revoked signatures, as illustrated in Table 1. Actually, this table shows a clear imbalance between the different groups as soon as the revocation list contains dozens of elements, a threshold that should be quickly reached in most scenarios. In those cases, we note that trying to stick to the minimal bound for q (thus decreasing the complexity of operations in \mathbb{G}_1), even if it significantly deteriorates the computational cost of a pairing, is a worthwhile goal as there is only one pairing to compute against roughly $6n$ exponentiations to generate or to verify the signature. This is all the more true since this pairing is only computed by the verifier, an entity that is assumed, in the context of DAA, to be much more powerful than the signer (usually a constrained device). To put it differently, we here need a curve that will optimize operations in \mathbb{G}_1 even if it is at the cost of a much more expensive pairing.

This scenario illustrates the limits of the global strategy for selecting parameters. The mainstream curves do not seem suitable here and we can hope for dramatic performance improvements by using a tailored curve. And this is not an isolated case as we explain in Sect. 3.

Our Contribution. The contribution of our paper is twofold. First, we investigate a different approach for selecting curve parameters. We indeed believe that standard families of curves, like BLS12, do not fit all cryptographic protocols and in particular impose a tradeoff (between the complexities of the different groups and operations) that is irrelevant in many contexts. Realizing that the era of optimality using Barreto-Naehrig curves is over, we choose to focus on the family of cryptographic protocols whose practicality depends on the implementation of the group \mathbb{G}_1 . This family is quite large because cryptographers

usually try to avoid as much as possible the other groups (\mathbb{G}_2 and \mathbb{G}_T) as the latter are much less efficient and even incompatible with some constrained devices (see *e.g.* [6]). We then look for curves with minimal \mathbb{G}_1 scalar multiplication complexity, which leads us to the case of prime embedding degree k , a setting that has been overlooked for a long time despite being immune to Kim's and Barbulescu's attacks [32, 33] mentioned above. We provide a security assessment, some benchmarks and a complexity evaluation of some curves from this setting that we compare to the most known alternatives. Our results show that the computational complexity of the operations on the first pairing group is 65% higher for the standard curves (with composite embedding degrees) comparing to our proposal (with prime embedding degrees), while yielding elements in \mathbb{G}_1 that are 20% larger. Of course, those composite-embedding-degree curves come at the cost of a less expensive pairing but our analysis shows that this overhead is acceptable in the context we consider.

Based on these results we investigate new curves that would match our need. We find a new one that goes one step further in the quest for optimizing the performance of \mathbb{G}_1 . We call this new curve, which constitutes our second contribution, BW19-P286 as it was generated using the Brezing-Weng strategy [11], has embedding degree 19 and is defined over a 286-bit field \mathbb{F}_q . The use of such a small q , which is close to the optimal bound 2^{256} , is particularly interesting for constrained devices (more specifically those with 32-or less- bits of architecture) as it reduces the number of machine-words compared to the state-of-the-art.

In the end, our results show that Kim's and Barbulescu's attacks do not necessarily imply a large increase of the complexity of pairing-based protocols that would in particular rule out the latter for constrained devices. On the contrary, we prove that we can retain the original efficiency for some parties. Of course, this is done to the detriment of the other parties but we argue that there are few use-cases where all entities are equally powerful. We nevertheless do not claim that our curves fit all contexts and in particular we do believe that standard curves still remain relevant, in particular when a large number of pairings is to be computed.

Roadmap. In Sect. 3 we describe some examples that justify our strategy for selecting curves. In Sect. 4 we outline the strategy to assess the cost of the Discrete Logarithm Problem and the security of our curves and provide two tailored curves in Sect. 5. Finally, we compared an implementation of the proposed curves with other curves in Sect. 6.

2 Preliminaries

Let $q > 3$ be a prime number. The field having q elements is noted \mathbb{F}_q and, for any $n > 1$, the extension field having q^n elements is noted \mathbb{F}_{q^n} . When we compute discrete logarithms in \mathbb{F}_{q^n} , we mean solving the Discrete Logarithm Problem in the group $(\mathbb{F}_{q^n} \setminus \{0\}, \times)$.

2.1 Elliptic Curves

An elliptic curve E/\mathbb{F}_q is the set of points (x, y) satisfying $y^2 = x^3 + ax + b$, where $a, b \in \mathbb{F}_q$ and $4a^3 + 27b^2 \neq 0$, enlarged with another point ∞ , called point at infinity. This equation is called the Short Weierstrass Model of the curve. For integer $n \geq 1$, the set of point $(x, y) \in (\mathbb{F}_{q^n})^2$ on E is noted $E(\mathbb{F}_{q^n})$. The set $E(\mathbb{F}_{q^n})$ can be equipped with a commutative internal law, with ∞ as identity element. We follow the convention of cryptographic literature and denote this group multiplicatively. For an integer ℓ coprime to q , we note $E(\mathbb{F}_{q^n})[\ell]$ the subgroup of $E(\mathbb{F}_{q^n})$ formed by points of order dividing ℓ , *i.e.* all points g such that $g^\ell = \infty$. The group $E(\mathbb{F}_{q^n})[\ell]$ is called the ℓ -torsion over \mathbb{F}_{q^n} . When we compute discrete logarithms in $E(\mathbb{F}_{q^n})[\ell]$, we mean solving the Discrete Logarithm Problem in the group $(E(\mathbb{F}_{q^n})[\ell], \cdot)$, *i.e.* if h is a power of g , find x such that $h = g^x$. There is a minimal $k \geq 1$ such that $E(\mathbb{F}_{q^k})[\ell]$ is isomorphic to $(\mathbb{Z}/\ell\mathbb{Z})^2$, this integer is called the embedding degree of q (with respect to ℓ), it is the order of $q \pmod{\ell}$.

The Frobenius endomorphism is defined as $(x, y) \mapsto (x^q, y^q) \in \text{End}(E)$. Its minimal polynomial is $X^2 - tX + q$, where t is aptly called the trace of the Frobenius, and the number of \mathbb{F}_q -rational points is $q - t + 1$. The discriminant Δ of that polynomial is $\Delta = t^2 - 4q$. We can always write $\Delta = Df^2$, where $D < 0$ is square-free. D is called the Complex Multiplication discriminant.

When the CM discriminant is small enough, there exist an endomorphism ϕ easily computable and an integer $\lambda > 0$ such that $\phi(g) = g^\lambda$ for all $g \in E(\mathbb{F}_q)[\ell]$. The main advantage of using the endomorphism ϕ is to roughly halve the computational cost of an exponentiation in $E(\mathbb{F}_q)[\ell]$ as evaluating ϕ is much more efficient than directly raising to the power λ . This is called the GLV method [24]. Suppose we want to compute g^a for a point $g \in E(\mathbb{F}_q)[\ell]$ and a scalar $a \pmod{\ell}$. We proceed as follow: compute a_0 and a_1 such that $a = a_0 + a_1\lambda$ (*e.g.* the Euclidean division of a by λ) and compute $g^{a_0} \cdot \phi(g)^{a_1}$. The result is g^a :

$$g^{a_0} \cdot \phi(g)^{a_1} = g^{a_0} \cdot g^{a_1\lambda} = g^{a_0+a_1\lambda} = g^a.$$

The size of a_0 and a_1 is expected to be half the size of ℓ as λ is a root of a degree 2 polynomial. The point $\phi(P)$ can be precomputed (if needed) and $g^{a_0} \cdot \phi(g)^{a_1}$ can be computed with any multi-exponentiation algorithm.

2.2 Bilinear Groups

Pairing-based cryptographic protocols consider a setting defined by three cyclic groups, \mathbb{G}_1 , \mathbb{G}_2 , and \mathbb{G}_T , of prime order ℓ (with respective identity element $1_{\mathbb{G}_1}$, $1_{\mathbb{G}_2}$ and $1_{\mathbb{G}_T}$), along with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with the following properties:

1. for all $g \in \mathbb{G}_1, \tilde{g} \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}/\ell\mathbb{Z}$, $e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{a \cdot b}$;
2. for any $g \neq 1_{\mathbb{G}_1}$ and $\tilde{g} \neq 1_{\mathbb{G}_2}$, $e(g, \tilde{g}) \neq 1_{\mathbb{G}_T}$;
3. the map e is efficiently computable.

As all these groups are of the same prime order, we know that there exist non-trivial homomorphisms $\varphi_1 : \mathbb{G}_1 \rightarrow \mathbb{G}_2$ and $\varphi_2 : \mathbb{G}_2 \rightarrow \mathbb{G}_1$. However, the latter may not be efficiently computable, which has a strong impact on cryptographic protocols and more specifically on the underlying computational assumptions. Following Galbraith, Paterson and Smart [23], this has led cryptographers to distinguish types of pairings: type-1, where both φ_1 and φ_2 are efficiently computable; type-2, where only φ_2 is efficiently computable; and type-3, where no efficiently computable homomorphism exists between \mathbb{G}_1 and \mathbb{G}_2 , in either direction. All these types can be instantiated with elliptic curves but type-3 pairings are preferred in practice both for their efficiency and their ability to support some useful cryptographic assumptions, *e.g.* decisional Diffie-Hellman in groups \mathbb{G}_1 and \mathbb{G}_2 .

We also note that it is possible to consider bilinear groups of composite order. However, prime order bilinear groups are much more efficient [26] and can actually emulate most features of their composite-order counterparts [21].

Usually, when bilinear groups are instantiated over an elliptic curve, the curve is ordinary (*i.e.* $t \not\equiv 0 \pmod{q}$), the number of \mathbb{F}_q -rational points $q - t + 1$ is a multiple of ℓ but not of ℓ^2 and pairing groups are taken as $\mathbb{G}_1 = E(\mathbb{F}_q)[\ell]$, $\mathbb{G}_2 \subset E(\mathbb{F}_{q^k})[\ell] \setminus \mathbb{G}_1$ and $\mathbb{G}_T \subset \mathbb{F}_{q^k}$, where k is the order of $q \pmod{\ell}$.

That will be our case here.

3 Schemes with Numerous Computations in \mathbb{G}_1

Before providing details on the way we select elliptic curve parameters, we elaborate on the motivation of our work, namely the benefits of selecting such parameters based on the characteristics of the cryptographic protocols. We are more specifically interested in the family of cryptographic protocols whose complexity essentially depends on the efficiency of \mathbb{G}_1 . This family may include protocols requiring to perform many exponentiations in \mathbb{G}_1 , as is the case with the EPID scheme we discuss in the introduction, but also schemes where the most constrained entity only has to compute operations in \mathbb{G}_1 , as in Direct Anonymous Attestation. These two primitives are today massively used in industrial products [2, 43] and are thus meaningful examples of this family of cryptographic protocols. To illustrate that the latter is not restricted to authentication algorithms we will also consider the case of two cryptographic accumulators that would benefit from the tailored curves we propose in our paper.

Table 1 highlights the specific need of two different anonymous authentication schemes [6, 13] that are, to our knowledge, the most efficient of their kind. For [13], we use the proof of non-revocation described by the same authors in [14]. We note that most alternatives and variants (*e.g.* [37] for group signature) present similar features so our conclusions also apply to them. Table 1 shows that the size of the signature only depends on the one of \mathbb{G}_1 elements (and on ℓ) and that the signer only has to perform operations in \mathbb{G}_1 . There are few pairings and operations in \mathbb{G}_2 to compute and only on the verifier side, which

is usually considered as more powerful than the signer in those contexts. Cryptographic protocols with such features are thus a good incentive for designing curves with efficient computations/elements in \mathbb{G}_1 .

Table 1. Complexity of some anonymous authentication schemes. \mathbf{e}_i refers to an exponentiation in \mathbb{G}_i and \mathbf{P} to a pairing computation, n is the number of revoked signatures.

Primitive	Ref.	Signature elements	Operation counts	
			Sign	Verify
DAA	[6]	$5\mathbb{G}_1 + 2\mathbb{Z}/\ell\mathbb{Z}$	$6\mathbf{e}_1$	$4\mathbf{e}_1 + 3\mathbf{P}$
EPID	[13]	$3\mathbb{G}_1 + 6\mathbb{Z}/\ell\mathbb{Z}$ $+ n(3\mathbb{G}_1 + \mathbb{Z}/\ell\mathbb{Z})$	$3\mathbf{e}_1 + 4\mathbf{e}_T$ $+ 6n\mathbf{e}_1$	$4\mathbf{e}_1 + 2\mathbf{e}_2 + 4\mathbf{e}_T$ $+ 1\mathbf{P} + 6n\mathbf{e}_1$

In Table 2, we consider two pairing-based accumulator schemes [15, 22]. We recall that the point of an accumulator system is to project a large number of elements into a single short value, called the accumulator. Additionally, for each of these elements, it is possible to generate a short evidence, called witness, that the element has indeed been accumulated. In practice, there are essentially two kinds of entities, the one that needs to prove that an element has been accumulated (by computing the corresponding witness) and the one that checks this proof. We will then divide the public parameters of such systems between the ones (\mathbf{pk}) necessary for the proof and the ones (\mathbf{vk}) necessary for the verification. Here again, Table 2 shows a clear asymmetry between the prover and the verifier. The former is only impacted by the performance of \mathbb{G}_1 and so would clearly benefit from a curve tailored to optimize this group. This is all the more true that in the applications considered in [15] and [22], the prover is usually a user’s device whereas the verifier is some service provider that can reasonably be considered as more powerful.

Table 2. Complexity of some accumulators schemes. The latter are called set commitment schemes in [22]. Here \mathbf{m}_1 refers to a group operation in \mathbb{G}_1 , n is a bound on the number of values to be accumulated and j is the number of values currently accumulated. The other notations are those from the previous table.

Ref.	Public parameters		Operation counts	
	\mathbf{pk}	\mathbf{vk}	Sign	Verify
[15]	$2n\mathbb{G}_1$	$n\mathbb{G}_2$	$j\mathbf{m}_1$	$2\mathbf{P}$
[22]	$n\mathbb{G}_1$	$n\mathbb{G}_2$	$(j - 1)\mathbf{e}_1$	$1\mathbf{e}_2 + 2\mathbf{P}$

4 Attacks Solving the DLP

Most cryptographic schemes using bilinear groups rely on problems that are easier than the Discrete Logarithm Problem (DLP). Unfortunately, the concrete hardness of these problems is not known so the common approach to generate bilinear groups is to select parameters that yield three groups \mathbb{G}_1 , \mathbb{G}_2 and \mathbb{G}_T where the DLP is believed to be hard. The latter problem has indeed been extensively studied over the last 40 years and several algorithms were proposed to solve it.

The DLP on elliptic curves is called ECDLP (EC stands for Elliptic Curve) and is considered the hardest discrete logarithm problem to solve as only generic algorithms [41] are known and used, such as Baby-Step-Giant-Step [40] or Pollard-rho [38]. Moreover, the Pohlig-Hellman method [36] reduces an instance of the DLP in a cyclic group of composite order n to several easier instances of the DLP in cyclic groups of order strictly dividing n . Hence, for efficiency and security reasons, the groups \mathbb{G}_1 and \mathbb{G}_2 must be of prime order, *i.e.* ℓ is a prime number. And since the best variant of Pollard-rho [9] compute a discrete logarithm in at most $\sqrt{\pi\ell/4} \approx 0.886\sqrt{\ell}$ steps, ℓ of 2λ bits is enough for a security of λ bits.

A well-known value, called ρ -value, is used to describe the efficiency of the representation of elements of \mathbb{G}_1 . It is computed as $\rho = \log(q)/\log(\ell)$ or as $\rho = \deg(q)/\deg(\ell)$ when q and ℓ are polynomial in $\mathbb{Q}[X]$. When $\rho = 1$, the curve is of prime order ℓ , this is the best case for the arithmetic efficiency on the curve (and so for \mathbb{G}_1).

While determining the size of \mathbb{G}_1 and \mathbb{G}_2 over the elliptic curve is pretty straightforward, doing the same for \mathbb{G}_T over the finite field \mathbb{F}_{q^k} is much harder! Indeed, discrete logarithms in \mathbb{F}_{q^k} are computed in sub-exponential time (and sometimes even in quasi-polynomial time) by Number Field Sieve (NFS) algorithms. In the general case, it is difficult to evaluate the complexity of the NFS algorithm, and of its variants (see [29] for more details). To give an idea of the time-complexity of NFS algorithms, we need to introduce the L -notation, which is defined by:

$$L_{q^k}[\alpha, c] = \exp((c + o(1))(\ln q^k)^\alpha (\ln \ln q^k)^{1-\alpha})$$

with $\alpha \in [0, 1]$ and $c > 0$. Intuitively, if $\alpha = 1$ then $L_{q^k}[\alpha, c]$ is exponential in $\log_2(q^k)$ whereas it is polynomial in $\log_2(q^k)$ if $\alpha = 0$. NFS-type algorithms (of our concern) all have time-complexity $L_{q^k}[1/3, c]$ for c ranging from $\sqrt[3]{32/9}$ to $\sqrt[3]{96/9}$. The constant c plays an important role in the concrete (*i.e.* not asymptotic) world, as between 20 and 30 bits of security can be lost for the same size of \mathbb{F}_{q^k} depending on the choices of q and k [4]. Two criteria determine which NFS-variant to use: whether q is special, *i.e.* if q is computed as the image of a polynomial of degree at least 3, and whether k is composite. This is summed up in Table 3.

Until recently, only curves with special q and composite k were considered as they offer the best performance for pairing computations. Barbulescu and

Table 3. How to choose the right NFS-variant.

	q not special	q special
k prime	NFS [39] ($9c^3 \in [64, 96]$)	SNFS [5] ($9c^3 \in [32, 64]$)
k composite	exTNFS [32] ($9c^3 \in [48, 64]$)	SexTNFS [32] ($9c^3 = 32$)

Duquesne updated key size estimations in [4]. For a 128-bit security level, they urge to use a finite field of size at least $k \log_2(q) = 2930$ (respectively 3618, 5004) if NFS (respectively exTNFS, SexTNFS) is the best algorithm for computing discrete logarithms in \mathbb{F}_{q^k} (the common practice was $k \log_2(q) = 3072$ [42]). To access curves' security, we will use the algorithm provided by Guillevic at <https://gitlab.inria.fr/t nfs-alpha/alpha>.

The only known methods for constructing curves with ρ close to 1 give the characteristic of the finite field as a polynomial (thus q is special), like Brezing-Weng constructions [11]; other general methods having ρ close to 2, like Cocks-Pinch constructions (see [28]). In this context, we will consider the recent bounds [3072, 5376] on the finite field size provided by Guillevic [27] to achieve 128 bits of security.

If we have $\log_2(\ell) = 256$ to satisfy 128 bits of security on the curve side, we then know that $\log_2(q) = 256\rho$ and the finite field \mathbb{F}_{q^k} has size $256\rho k$. Guillevic's bounds then give us:

$$3072 \leq 256\rho k \leq 5376,$$

which, together with the inequalities $1 \leq \rho \leq 2$, allows us to derive the set of potential values for k : $6 \leq k \leq 21$. Concretely, this means that there is no point in considering values $k < 6$ as they are incompatible with the targeted 128-bit security level (for the range of ρ -values we consider) and selecting a value $k > 21$ would be an overkill.

So far, we have just managed to derive some bounds on the different parameters of the curves. Unfortunately, as we explain above, there is no simple choice within these bounds as security and efficiency of the resulting bilinear groups may significantly differ from one set of parameters to another. In particular, there is no linearity in the security evaluation as, for instance, the security of \mathbb{F}_{q^k} is significantly higher in the “prime” cases $k \in \{11, 13\}$ than in the case $k = 12$. This means that we can select smaller q values (which improves performance of \mathbb{G}_1) in the former case. Unfortunately, a similar issue arises regarding efficiency of \mathbb{F}_{q^k} , but with opposite conclusions, as non-prime k (especially even ones) yield more efficient pairings and group operations. It is thus necessary to make a choice between these different parameters, in particular in the case of cryptographic protocols with unbalanced complexities, such as the ones we consider in Sect. 3. As the latter would benefit of fast \mathbb{G}_1 computations and short representations of its elements, we dedicate the next section to the selection of parameters that will optimize the performance of this group.

5 Curves Optimizing Operation in \mathbb{G}_1

The first group of the pairing \mathbb{G}_1 is defined as $E(\mathbb{F}_q)[\ell]$. We have an incentive to reduce the size of q as computations in \mathbb{G}_1 would be faster. To ensure security on the elliptic curve, we need $\log_2(q) \geq \log_2(\ell) \geq 256$. Thus q is at least a 5-machine-word on a 64-bit computer. Indeed, q has more than 257 bits, because a 256-bit field would imply $\rho = 1$ and, the biggest known $k \in [6, 21]$ for that ρ -value is 12, which does not ensure a 128-bit security in \mathbb{G}_T [4].

We would like to be able to use the GLV method [24], that is, our curves should have a small Complex Multiplication discriminant. When curves are chosen either in the form $y^2 = x^3 + ax$ with a primitive fourth root of unity in \mathbb{F}_q and CM discriminant -1 , or in the form $y^2 = x^3 + b$ with a primitive third root of unity in \mathbb{F}_q and CM discriminant -3 , the GLV-endomorphism is easy to write down and relatively cheap to compute.

As explained above, we cannot hope for better than 5 machine-words for q on 64-bit architecture. A 5-machine-word q means that $1 < \rho \leq 1.25$. Searching in the Taxonomy by Freeman, Scott and Teske [20], the curve we are looking for has embedding degree $k \in \{8, 11, 13, 16, 17, 19\}$.

The embedding degree 8 does not provide a secure finite field \mathbb{F}_{q^k} , as it is of at most $8 \times 1.25 \times 256 = 2560$ bits, and so is discarded. The embedding degree 16 corresponds to the well-known KSS family of pairing-friendly curves [31]. Barbulescu and Duquesne [4] state that q must be at least a 330-bit prime, *i.e.* a 6-machine-word prime, and they give such primes.

Thus we focus our search for curves from the taxonomy [20] having embedding degree $k \in \{11, 13, 17, 19\}$. All those curves correspond to Freeman, Scott and Teske Construction 6.6, which is a generalization of the Brezing and Weng construction [11]. It defines the prime q , the Frobenius trace t and the order ℓ as polynomials in $\mathbb{Q}[X]$, where $\ell(X)$ is a cyclotomic polynomial dividing both $q(X) - t(X) + 1$ and $q(X)^k - 1$.

5.1 Curves Over a Five-64-Bit-Machine-Word Prime Field

Construction 6.6 from [20] is different given the residue $k \pmod{6}$. Since we have $13 \equiv 19 \equiv 1 \pmod{6}$ and $11 \equiv 17 \equiv 5 \pmod{6}$, we only give the relevant cases of Construction 6.6.

In the case $k \equiv 1 \pmod{6}$, the prime $q(X)$, the Frobenius trace $t(X)$ and the order $\ell(X)$ are given as:

$$\begin{aligned} q(X) &= \frac{1}{3}(X + 1)^2(X^{2k} - X^k + 1) - X^{2k+1}, \\ t(X) &= -X^{k+1} + X + 1 \quad \text{and} \\ \ell(X) &= \Phi_{6k}(X), \end{aligned}$$

and in the case $k \equiv 5 \pmod{6}$, they are given as:

$$\begin{aligned} q(X) &= \frac{1}{3}(X^2 - X + 1)(X^{2k} - X^k + 1) + X^{k+1}, \\ t(X) &= X^{k+1} + 1 \quad \text{and} \\ \ell(X) &= \Phi_{6k}(X). \end{aligned}$$

Plugging in the different values of k gives Table 4. Note that to find a curve, we need to find a $x_0 \in \mathbb{Z}$ such that $\ell(x_0)$ and $q(x_0)$ are prime integers. We choose x_0 satisfying $x_0 \equiv 2 \pmod{3}$ so $q(x_0)$ is an integer. The last column of Table 4 is the search range of $\log_2(|x_0|)$ in $[256/\deg(\ell), 320/\deg(q)[$ so that $q(x_0)$ is a 5-machine-word integer and $\ell(x_0)$ is at least a 256-bit integer (for readability, the interval is given with rounded integer values).

Table 4. Parameters and search range for curves with $k \in \{11, 13, 17, 19\}$

k	$\deg(q)$	$\deg(\ell)$	ρ	$\log_2(x_0)$
11	24	20	1.20	$[12, 14[$
13	28	24	1.167	$[10, 12[$
17	36	32	1.125	$[8, 9[$
19	40	36	1.111	$[7, 8[$

After a computer search, we have found no solution for $k \in \{11, 17\}$. For $k = 13$, we found $x_0 = -2224$ and for $k = 19$, we found $x_0 = -145$. Inferring the naming convention used in relic-toolkit [3], the first curve is named BW13-P310 and the second one BW19-P286. As Construction 6.6 curves have CM discriminant -3 , both curves are of the form $y^2 = x^3 + b$.

Curve BW13-P310. Setting $k = 13$ into Construction 6.6 [20] gives:

$$\begin{aligned} q(X) &= \frac{1}{3}(X + 1)^2(X^{26} - X^{13} + 1) - X^{27}, \\ t(X) &= -X^{14} + X + 1 \quad \text{and} \\ \ell(X) &= \Phi_{78}(X). \end{aligned}$$

Plugging in $x_0 = -2224$ yields a $q(x_0)$ of 310 bits and a $\ell(x_0)$ of 267 bits, thus $\rho = 1.161$. The corresponding $\mathbb{F}_{q^{13}}$ is a 4027-bit finite field.

To look for a b , we increase the value of $|b|$, check that the number of points on the curve $y^2 = x^3 + b$ is equal to $q(x_0) - t(x_0) + 1$. We found $b = -17$. So we defined the curve BW13-P310 by the equation $y^2 = x^3 - 17$.

We also point out that the exact same curve has been given by Aurore Guillevic in [27]. She made a thorough security analysis and estimated that the cost of the DLP in the finite field $\mathbb{F}_{q^{13}}$ is 140 bits. Hence, the curve BW13-P310 has a security of at least 128 bits.

Curve BW19-P286. Setting $k = 19$ into Construction 6.6 [20] gives:

$$\begin{aligned} q(X) &= \frac{1}{3}(X + 1)^2(X^{38} - X^{19} + 1) - X^{39}, \\ t(X) &= -X^{20} + X + 1 \quad \text{and} \\ \ell(X) &= \Phi_{114}(X). \end{aligned}$$

Plugging in $x_0 = -145$ yields a $q(x_0)$ of 286 bits and a $\ell(x_0)$ of 259 bits, thus $\rho = 1.105$. The corresponding $\mathbb{F}_{q^{19}}$ is a 5427-bit finite field.

To look for a b , we do the same as before. The smallest $|b|$ is $b = 31$. So we defined the curve BW19-P286 by the equation $y^2 = x^3 + 31$. To our knowledge, this curve has never been proposed in the literature.

To evaluate the cost of the DLP in $\mathbb{F}_{q^{19}}$, we follow the work of Guillevic [27], the same she did for the previous curve BW13-P310. To find the curve BW19-P286 using Guillevic’s Algorithm 3.1 [27], we plugged in the parameters $k = 19$, $D = 3$, $e_0 = 13$ and use the substitution of indeterminate $X \mapsto -X$.

Before running the estimating program on our parameters, we applied Variant 4 [27] to the polynomial $q(-X)$, yielding a polynomial $Q(X)$ such that $Q(u^3) = 3q(-u)$, for $u = -x_0 = 145$ and

$$Q(X) = (u + 1)X^{13} + u^2X^{12} + X^7 + u(1 - 2u)X^6 + u^2 - 2u + 1.$$

Then we obtain that the cost of the DLP in $\mathbb{F}_{q^{19}}$ is 160 bits, thus providing BW19-P286 with a security of at least 128 bits.

5.2 GLV Endomorphism on BW13-P310 and BW19-P286

As stated in the preliminaries, the discriminant of the minimal polynomial of the Frobenius endomorphism can be written as $Df^2 = t^2 - 4q$, where $D < 0$ is the CM discriminant and t is the trace of the Frobenius. The endomorphism $\phi : (x, y) \mapsto (\omega x, y)$, with ω a primitive third root of unity, corresponds to an exponentiation $(\omega x, y) = (x, y)^\lambda$ in $\mathbb{G}_1 = E(\mathbb{F}_q)[\ell]$ for q and ℓ distinct primes.

For both our curves, the CM discriminant is $D = -3$, as a consequence we have $4q = t^2 + 3f^2$. Since $\omega \in \mathbb{F}_q$ is a primitive third root of unity, ω satisfies $\omega^2 + \omega + 1 = 0$. We can take $\omega = (\sqrt{-3} - 1)/2$, where $\sqrt{-3} \equiv t/f \pmod{q}$. Thus

$$\omega \equiv \frac{t - f}{2f} \pmod{q}.$$

Similarly, since $\phi^3 = \text{id}_E$ in $\text{End}(E)$, we know that $\lambda \in \mathbb{Z}/\ell\mathbb{Z}$ satisfies the equation $\lambda^2 + \lambda + 1 = 0$, *i.e.* it can also be taken as $(\sqrt{-3} - 1)/2$. However, here, $\sqrt{-3} \equiv (t - 2)/f \pmod{\ell}$. Indeed, ℓ divides the number of points on the curve, so $q \equiv t - 1 \pmod{\ell}$ and $4(t - 1) \equiv t^2 + 3f^2 \pmod{\ell}$. Thus

$$\lambda \equiv \frac{t - f - 2}{2f} \pmod{\ell}.$$

Note that in practice, adjustments may be needed as $(\omega x, y) = (x, y)^{\pm\lambda}$ or $(\omega^2 x, y) = (x, y)^{\pm\lambda}$. In the case of BW13-P310, λ has bit-length 146, whereas it is only 137 in the case of BW19-P286.

Comments on BW13-P310 and BW19-286. We provide in the next section several benchmarks to compare our new curve with BW13-P310 but also with other curves from popular families. However, we can already note that BW13-P310 and BW19-P286 clearly match our strategy of optimizing the group \mathbb{G}_1 to the detriment of the other groups. In this respect, our new curve BW19-P286 goes one step further than BW13-P310 by reducing the size of q by roughly 25 bits. This difference is significantly amplified in the context of constrained devices as it results in less machine words. Indeed, even for 32-bit architecture, BW19-P286 yields a prime q with one less machine-word than BW13-P310, which clearly impacts performance, as illustrated below.

6 Implementation and Comparison

We implemented the \mathbb{G}_1 arithmetic of both curves using relic-toolkit [3], and made some comparison with other curves already implemented in relic-toolkit and aiming the 128-bit security.

The curves selected from the framework are BN-P446, a Barreto-Naehrig curve [8] over a 446-bit prime field; K16-P339, a Kachisa-Schaefer-Scott curve [31] of embedding degree 16 over a 339-bit field; B12-P446, a Barreto-Lynn-Scott curve [7] of embedding degree 12 over a 446-bit field; and CP8-P544, a Cocks-Pinch curve [28] of embedding degree 8 over a 544 prime field. We chose the last curve as it is coming from recent works [27, 28] that promote them for the 128-bit security level. We included a BN, BLS and KSS curves in our table as those families of curves are well-known and were updated by Barbulescu and Duquesne [4]. Also note that, setting aside our curves, only the curve K16-P339 was implemented by us in the framework.

All curves enjoy GLV-endomorphisms which were already implemented or have been implemented.

6.1 Operation in \mathbb{G}_1

In Table 5 we compare the cost of one exponentiation in the group \mathbb{G}_1 by compiling the relic-toolkit either for x64 architecture or for x86 architecture with a word size of 32 bits. Times are given in microseconds (the number of iterations was 10^6) and computations were done on a laptop equipped with a Intel Core i7-6600u CPU at 2.60 GHz.

This table shows a clear relation, almost quadratic, between complexity and the number of words necessary to represent q . It also highlights the downside of Barreto-Naehrig curves that generate elliptic curves of prime order $\ell \sim q$. Indeed, what was considered as an advantage (prime order curves make group membership tests trivial) turns out to be a strong limitation as it forces ℓ to grow

Table 5. Benchmark for one exponentiation in \mathbb{G}_1 . For both architectures, row “words” indicates the number of machine words used to store the prime q , row “time” indicates the obtained computation time of one exponentiation in \mathbb{G}_1 rounded to the nearest microsecond ($1 \mu\text{s} = 10^{-6}$ s), and row “time ratio” indicates the time ratio between the curve in the column and the curve **BW19-P286**.

Curve		BW19-P286	BW13-P310	K16-P339	B12-P446	BN-P446	CP8-P544
64-bit	words	5	5	6	7	7	9
	time (μs)	293	304	482	611	855	1058
	time ratio	1	1.04	1.65	2.09	2.92	3.61
32-bit	words	9	10	11	14	14	17
	time (μs)	1010	1220	1664	2510	3600	4180
	time ratio	1	1.21	1.65	2.49	3.56	4.14

unnecessarily. This negatively impacts both exponentiation (as the exponents are roughly 75% greater than those of the other curves) and the size of scalars.

In all cases, this table shows that our curve **BW19-P286** offers the best performance for \mathbb{G}_1 , in particular for architecture smaller than 64 bits. It at least halves the complexity of exponentiations in \mathbb{G}_1 compared to mainstream curves such as **B12-P446** and also significantly decreases the size of group elements, which clearly fits the needs of some cryptographic protocols such as the ones we presented in Sect. 3.

6.2 Operation in \mathbb{G}_2

The operations in \mathbb{G}_2 are unfortunately the ones that are the most impacted by our choice of prime embedding degree. Indeed, **BW13-P310** and **BW19-P286** have \mathbb{G}_2 defined over $\mathbb{F}_{q^{13}}$ and $\mathbb{F}_{q^{19}}$ respectively, whereas **B12-P446**, **BN-P446** and **CP8-P544** all have \mathbb{G}_2 defined over \mathbb{F}_{q^2} thanks to quartic or sextic twists, and **K16-P339** has \mathbb{G}_2 defined over \mathbb{F}_{q^4} thanks to a quartic twist. As all curves are usually expressed with the same model using the same system of coordinates, only the cost of the multiplication in the extension impacts the cost of the operations in \mathbb{G}_2 . Using a Karatsuba-like implementation, the multiplication in \mathbb{F}_{q^k} is roughly $k^{\log_3 2}$ times as expensive as the one in \mathbb{F}_q .

6.3 Pairing Computation

The computation of the pairing is usually split between two parts: the evaluation of the Miller loop and the final exponentiation. Here we give computation for a multiple of the Optimal Ate Pairing [44], since we picked the final exponentiation from Kim, Kim and Cheon [34]. The values in Table 6 are from [27, 28], completed with the ones from below.

Let \mathbf{m}_k , \mathbf{s}_k , \mathbf{i}_k , \mathbf{f}_k respectively denote a multiplication, a square, an inversion, a Frobenius map (*i.e.* the q -th power map) over \mathbb{F}_{q^k} . We drop the index when the operation is over \mathbb{F}_q (*e.g.* $\mathbf{m} = \mathbf{m}_1$). As $k \in \{13, 19\}$ is prime, we estimate

$\mathbf{m}_k = \mathbf{s}_k = k^{\log_2 3} \mathbf{m}$ with a Karatsuba-like implementation and $\mathbf{f}_k = (k - 1)\mathbf{m}$ as in [28].

Miller Loop. Using Equation (7) from [27], Guillevic gives a lower bound on the cost of the Miller loop. For both BW13-P310 and BW19-P286, the optimal ate Miller loop has length $u^2 + up + p^2$, as it is a multiple of ℓ [44].

For BW13-P310, the Miller loop has length $u^2 + up + p^2$, where $u = 2224$ is a 12-bit integer with Hamming weight 4 and p is a 310-bit prime. From her Equation (7) [27], Guillevic obtains $949\mathbf{m} + 313\mathbf{m}_{13} + 177\mathbf{s}_{13} + 5\mathbf{f}_{13} + 2\mathbf{i}_{13}$. Substituting $\mathbf{m}_{13} = \mathbf{s}_{13} = 59\mathbf{m}$ and $\mathbf{f}_{13} = 12\mathbf{m}$ in that formula yields a lower bound on the cost of the optimal ate Miller loop, *i.e.* $29919\mathbf{m} + 2\mathbf{i}_{13}$.

For BW19-P286, the length of the Miller loop is $u^2 + up + p^2$, where $u = 145$ is a 8-bit integer with Hamming weight 3 and p is a 286-bit prime. From the same equation as Guillevic [27], we obtain $912\mathbf{m} + 212\mathbf{m}_{19} + 115\mathbf{s}_{19} + 5\mathbf{f}_{19} + 2\mathbf{i}_{19}$. Substituting $\mathbf{m}_{19} = \mathbf{s}_{19} = 107\mathbf{m}$ and $\mathbf{f}_{19} = 18\mathbf{m}$ in that formula yields a lower bound on the cost of the optimal ate Miller loop, *i.e.* $35991\mathbf{m} + 2\mathbf{i}_{19}$.

Final Exponentiation. As usual, the final exponentiation $(q^k - 1)/\ell$ of the Optimal Ate Pairing is split between an easy part $(q^k - 1)/\Phi_k(q)$ and a hard part $\Phi_k(q)/\ell$. Since $k \in \{13, 19\}$ is prime, the easy part is simply $q - 1$, costing $\mathbf{f}_k + \mathbf{i}_k$. For the hard part, Kim, Kim and Cheon [34] noticed that $\Phi_k(q)/\ell$ can be decompose in base q to make use of the Frobenius and the coefficients can be reduced by looking for a short vector in a specifically designed lattice. However, instead of raising to the power $\Phi_k(q)/\ell$, this method [34] raises to a multiple power $m\Phi_k(q)/\ell$.

More precisely, they write

$$m \frac{\Phi_k(q)}{\ell} = \sum_{i=0}^{k-2} a_i q^i$$

and find the $k - 1$ coefficients $(a_i)_{0 \leq i \leq k-2}$ as the shortest vector in the $\dim(k - 1)$ lattice spanned by the lines of the following matrix:

$$\begin{bmatrix} \frac{\Phi_k(q)}{\ell} & 0 & 0 & \dots & 0 \\ -q & 1 & 0 & \dots & 0 \\ -q^2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & & & \\ -q^{k-2} & 0 & \dots & 0 & 1 \end{bmatrix}.$$

Then, they compute the Frobenius of the element they want to exponent, up to the $(k - 1)$ -th q -power, costing $(k - 2)\mathbf{f}_k$.

If the exponents a_i 's were longer, we would have needed $(2^{k-1} - k)\mathbf{m}_k$ to compute all combinations of $(k - 1)$ Frobenius powers. However, we do not use all of these combinations, only roughly $\mathcal{O}(\log_2 q)$ of them. Finally the length of the

multi-exponent is $\max_i \{\lfloor \log_2 a_i \rfloor\}$, resulting in an average final exponentiation costing

$$(k - 1)\mathbf{f}_k + (\mathcal{O}(\log_2 q) + \max_i \{\lfloor \log_2 a_i \rfloor\})\mathbf{m}_k + \max_i \{\lfloor \log_2 a_i \rfloor\}\mathbf{s}_k + \mathbf{i}_k,$$

omitting some inversion due to the sign of some a_i 's.

For BW13-P310, the value of $\max_i \{\lfloor \log_2 a_i \rfloor\}$ is 287 and 8 of the 12 a_i 's are negative. Only 191 different combinations of Frobenius powers are used and it costs $341\mathbf{m}_{13}$ to compute them. Also, there are 5 positions (in the binary expansion) where all the a_i 's have their bit set to 0, resulting in no multiplication at those positions for the multi-exponentiation, that thus requires $282\mathbf{m}_{13} + 287\mathbf{s}_{13}$. Combining everything yields an final exponentiation cost of $12\mathbf{f}_{13} + 623\mathbf{m}_{13} + 287\mathbf{s}_{13} + 9\mathbf{i}_{13}$, *i.e.* $53\ 834\mathbf{m} + 9\mathbf{i}_{13}$.

For BW19-P286, the value of $\max_i \{\lfloor \log_2 a_i \rfloor\}$ is 271 and 12 of the 18 a_i 's are negative. Only 222 different combinations of Frobenius powers are used and it costs $1028\mathbf{m}_{19}$ to compute them. The multi-exponentiation requires $271(\mathbf{m}_{19} + \mathbf{s}_{19})$. Combining everything yields an final exponentiation cost of $18\mathbf{f}_{19} + 1299\mathbf{m}_{19} + 271\mathbf{s}_{19} + 13\mathbf{i}_{19}$, *i.e.* $160\ 824\mathbf{m} + 13\mathbf{i}_{19}$.

Table 6. Operation count for Miller loop (M.), final exponentiation (E.) and total pairing (T.). Inversions over odd-degree extension field are displayed as they are more costly than over an even-degree one (those usually cost one conjugaison over a quadratic subfield).

	BW19-P286	BW13-P310	K16-P339	B12-P446	BN-P446	CP8-P544
M.	$35991\mathbf{m} + 2\mathbf{i}_{19}$	$29919\mathbf{m} + 2\mathbf{i}_{13}$	$7691\mathbf{m}$	$7805\mathbf{m}$	$11620\mathbf{m}$	$4502\mathbf{m}$
E.	$160824\mathbf{m} + 13\mathbf{i}_{19}$	$53834\mathbf{m} + 9\mathbf{i}_{13}$	$18235\mathbf{m}$	$7723\mathbf{m}$	$5349\mathbf{m}$	$7056\mathbf{m}$
T.	$196815\mathbf{m} + 15\mathbf{i}_{19}$	$83753\mathbf{m} + 11\mathbf{i}_{13}$	$25926\mathbf{m}$	$15528\mathbf{m}$	$16969\mathbf{m}$	$11558\mathbf{m}$

From Table 6, the cost of the pairing for BW19-P286 is roughly 12 times higher than the one for B12-P446. However, doing the benchmark on both finite field gives a multiplication twice faster on the 286-bit finite field (90 ns) than the 446-bit one (190 ns). Hence, we estimate that the pairing over BW19-P286 is 6 times slower than the pairing over B12-P446.

7 Conclusion

In this paper, we have given an incentive to change the way pairing-friendly elliptic curve are constructed by shifting the optimization away from the balance between all operations (group exponentiation and pairing) towards only some operations (that might be used by constrained entities involved in cryptographic protocols).

Thus, we focused on elliptic curves with a fast exponentiation in the first pairing group upon noticing that the instantiation of some cryptographic protocols, *e.g.* Group Signature-like schemes, would benefit from such curves.

Along the way, we have described a new curve that is particularly relevant for cryptographic protocols extensively using exponentiation in the first pairing group. That curve is twice faster in that group and its pairing computation is reasonably six times slower compared to a BLS curve over a 446-bit field.

We leave to future work to investigate other protocol-curve dependencies.

Acknowledgements. The authors are grateful for the support of the ANR through projects ANR-19-CE39-0011-04 PRESTO and ANR-18-CE-39-0019-02 MobiS5.

References

1. Abe, M., Hoshino, F., Ohkubo, M.: Design in type-I, run in type-III: fast and scalable bilinear-type conversion using integer programming. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 387–415. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_14
2. Allée, G.: EPID for IoT Identity (2016). https://img.en25.com/Web/McAfeeE10BuildProduction/a6dd7393-63f8-4c08-b3aa-89923182a7e5_EPID_Overview_Public_2016-02-08.pdf
3. Aranha, D.F., Gouvêa, C.P.L.: RELIC is an Efficient LIBrary for Cryptography. <https://github.com/relic-toolkit/relic>
4. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. *J. Cryptol.* **32**, 1298–1336 (2019)
5. Barbulescu, R., Gaudry, P., Guillevic, A., Morain, F.: Improving NFS for the discrete logarithm problem in non-prime finite fields. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 129–155. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_6
6. Barki, A., Desmouhins, N., Gharout, S., Traoré, J.: Anonymous attestations made practical. In: ACM WiSec 2017, pp. 87–98. ACM (2017)
7. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: Cimato, S., Persiano, G., Galdi, C. (eds.) SCN 2002. LNCS, vol. 2576, pp. 257–267. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36413-7_19
8. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006). https://doi.org/10.1007/11693383_22
9. Bernstein, D.J., Lange, T., Schwabe, P.: On the correct use of the negation map in the Pollard rho method. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 128–146. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19379-8_8
10. Bichsel, P., Camenisch, J., Neven, G., Smart, N.P., Warinschi, B.: Get shorty via group signatures without encryption. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 381–398. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15317-4_24
11. Brezing, F., Weng, A.: Elliptic curves suitable for pairing based cryptography. *Des. Codes Cryptogr.* **37**, 133–141 (2005)

12. Brickell, E.F., Camenisch, J., Chen, L.: Direct anonymous attestation. In: ACM CCS 2004, pp. 132–145. ACM (2004)
13. Brickell, E., Li, J.: Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. *Int. J. Inf. Priv. Secur. Integr.* **2**(1), 3–33 (2011). IEEE Computer Society, In IEEE SocialCom
14. Brickell, E., Li, J.: Enhanced privacy ID: a direct anonymous attestation scheme with enhanced revocation capabilities. *IEEE Trans. Dependable Secur. Comput.* **9**(3), 345–360 (2012)
15. Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 481–500. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00468-1_27
16. Chatterjee, S., Menezes, A.: Type 2 structure-preserving signature schemes revisited. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 286–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_13
17. Cheon, J.H., Han, K., Lee, C., Ryu, H., Stehlé, D.: Cryptanalysis of the CLT13 multilinear map. *J. Cryptol.* **32**, 547–565 (2019)
18. Cheon, J.H., Lee, C., Ryu, H.: Cryptographic multilinear maps and their cryptanalysis. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **101**, 12–18 (2018)
19. Coron, J.-S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 476–493. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_26
20. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. *J. Cryptol.* **23**, 224–280 (2010)
21. Freeman, D.M.: Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 44–61. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_3
22. Fuchsbauer, G., Hanser, C., Slamanig, D.: Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *J. Cryptol.* **32**(2), 498–546 (2019)
23. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. *Discret. Appl. Math.* **156**, 3113–3121 (2008)
24. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_11
25. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 1–17. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_1
26. Guillemic, A.: Comparing the pairing efficiency over composite-order and prime-order elliptic curves. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 357–372. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38980-1_22
27. Guillemic, A.: A short-list of pairing-friendly curves resistant to special TNFS at the 128-bit security level. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12111, pp. 535–564. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45388-6_19

28. Guillevic, A., Masson, S., Thomé, E.: Cocks-pinch curves of embedding degrees five to eight and optimal ate pairing computation. *Des. Codes Cryptogr.* **88**(6), 1–35 (2020)
29. Guillevic, A., Morain, F.: Discrete Logarithms. In: *Guide to Pairing-Based Cryptography*. CRC Press - Taylor and Francis Group (2016)
30. Joux, A.: A one round protocol for tripartite Diffie-Hellman. *J. Cryptol.* **17**, 263–276 (2004)
31. Kachisa, E.J., Schaefer, E.F., Scott, M.: Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field. In: Galbraith, S.D., Paterson, K.G. (eds.) *Pairing 2008*. LNCS, vol. 5209, pp. 126–135. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85538-5_9
32. Kim, T., Barbulescu, R.: Extended tower number field sieve: a new complexity for the medium prime case. In: Robshaw, M., Katz, J. (eds.) *CRYPTO 2016*. LNCS, vol. 9814, pp. 543–571. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_20
33. Kim, T., Jeong, J.: Extended tower number field sieve with application to finite fields of arbitrary composite extension degree. In: Fehr, S. (ed.) *PKC 2017*. LNCS, vol. 10174, pp. 388–408. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54365-8_16
34. Kim, T., Kim, S., Cheon, J.H.: On the final exponentiation in Tate pairing computations. *IEEE Trans. Inf. Theory* **59**(6), 4033–4041 (2013)
35. Menezes, A., Vanstone, S.A., Okamoto, T.: Reducing elliptic curve logarithms to logarithms in a finite field. In: *ACM STOC* (1991)
36. Pohlig, S., Hellman, M.: An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance (Corresp.). *IEEE Trans. Inf. Theory* **24**, 106–110 (1978)
37. Pointcheval, D., Sanders, O.: Short randomizable signatures. In: Sako, K. (ed.) *CT-RSA 2016*. LNCS, vol. 9610, pp. 111–126. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8_7
38. Pollard, J.M.: Monte Carlo methods for index computation (mod p). *Math. Comput.* **32**, 918–924 (1978)
39. Sarkar, P., Singh, S.: New complexity trade-offs for the (multiple) number field sieve algorithm in non-prime fields. In: Fischlin, M., Coron, J.-S. (eds.) *EUROCRYPT 2016*. LNCS, vol. 9665, pp. 429–458. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_17
40. Shanks, D.: Class number, a theory of factorization, and genera. In: *1969 Number Theory Institute*, pp. 415–440. American Mathematical Society (1971)
41. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) *EUROCRYPT 1997*. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_18
42. Smart, N.P.: Algorithms, key size and protocols report, *ECRYPT - CSA* (2018). <http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf>
43. TCG (2015). <https://trustedcomputinggroup.org/authentication/>
44. Vercauteren, F.: Optimal pairings. *IEEE Trans. Inf. Theory* **56**, 455–461 (2010)



Revisiting ECM on GPUs

Jonas Wloka¹✉, Jan Richter-Brockmann², Colin Stahlke³,
Thorsten Kleinjung⁴, Christine Priplata³, and Tim Güneysu^{1,2}

¹ DFKI GmbH, Cyber-Physical Systems, Bremen, Germany
jonas.wloka@dfki.de

² Ruhr University Bochum, Horst Görtz Institute Bochum, Bochum, Germany
{jan.richter-brockmann,tim.gueneysu}@rub.de

³ CONET Solutions GmbH, Hennef, Germany
{cstahlke,cpriplata}@conet.de

⁴ EPFL IC LACAL, Station 14, Lausanne, Switzerland
thorsten.kleinjung@epfl.ch

Abstract. Modern public-key cryptography is a crucial part of our contemporary life where a secure communication channel with another party is needed. With the advance of more powerful computing architectures – especially Graphics Processing Units (GPUs) – traditional approaches like RSA and Diffie-Hellman schemes are more and more in danger of being broken.

We present a highly optimized implementation of Lenstra’s ECM algorithm customized for GPUs. Our implementation uses state-of-the-art elliptic curve arithmetic and optimized integer arithmetic while providing the possibility of arbitrarily scaling ECM’s parameters allowing an application even for larger discrete logarithm problems. Furthermore, the proposed software is not limited to any specific GPU generation and is to the best of our knowledge the first implementation supporting multiple device computation. To this end, for a bound of $B_1 = 8192$ and a modulus size of 192 bit, we achieve a throughput of 214 thousand ECM trials per second on a modern RTX 2080 Ti GPU considering only the first stage of ECM. To solve the Discrete Logarithm Problem for larger bit sizes, our software can easily support larger parameter sets such that a throughput of 2 781 ECM trials per second is achieved using $B_1 = 50\,000$, $B_2 = 5\,000\,000$, and a modulus size of 448 bit.

Keywords: ECM · Cryptanalysis · Prime factorization · GPU

1 Introduction

Public-Key Cryptography (PKC) is a necessary part of any large-scale cryptographic infrastructure in which communicating partners are unable to exchange keys over a secure channel.

PKC systems use a keypair of public and private key, designed such that to retrieve the secret counterpart of a public key, a potential attacker would have to

solve a mathematically hard problem. Traditionally – most prominently in RSA and Diffie-Hellman schemes – *factorization of integers* or computing a *discrete logarithm* are the hard problems at the core of the scheme. For reasonable key sizes, both these problems are considered to be computationally infeasible for classical computers.

If built, large-scale quantum computers, are able to compute both factorization and discrete logarithms in polynomial time. However, common problem sizes are not only under threat by quantum computers: With Moore’s Law still mostly accurate, classical computing power becomes more readily available at a cheaper price. Additionally, in the last decade more diverse computing architectures are available. Graphics Processing Units (GPUs) have been used in multiple scientific applications – including cryptanalysis – for their massive amount of parallel computing cores. As the problem of factorization and computing a discrete logarithm can (in part) be parallelized, GPU architectures fit these tasks well.

Nowadays the NIST recommends to use 2048- and 3072-bit keys for RSA [3]. Factoring keys of this size is out of reach for current publicly known algorithms on classical computers. However, in [35], the authors found that still tens of thousands of 512-bit keys are used in the wild, which could be factored for around \$70 within only a few hours.

To find the prime factorization of large numbers, the currently best performing algorithm is the General Number Field Sieve (GNFS). During a run of the GNFS algorithm, many numbers smaller than the main target need to be factored which is commonly done by Lenstra’s Elliptic-Curve Factorization Method (ECM) and is inherently parallel.

Related Work. ECM has been implemented on graphic cards before and several approaches at optimizing the routines used in ECM on the different levels have already been published.

A general overview of factoring, solving the Discrete Logarithm Problem (DLP) and the role of ECM in such efforts is given in [26, 27]. The most recent result in factorization and solving a DLP was announced in December 2019 with the factorization of RSA-240 and with solving the DLP for a 795-bit prime [13]. Previous records for the factorization of a 768-bit RSA modulus and the computation of a 768-bit DLP are reported in [21, 22] and [23], respectively. A general overview of factorization methods, and the use of ECM on GPUs within the GNFS is given in [30].

The original publication of ECM by Lenstra in [28] has since received much attention, especially the choice of curves [5, 7] and parameters [17] was found to have a major impact on the algorithm’s performance. Optimizing curve arithmetic for parallel architectures – mostly GPUs – has been a topic of many scientific works as well [1, 2, 19, 25, 29]. A detailed description of a parallel implementation for GPUs is given in [11].

To this end, the implementation of ECM on GPUs has attracted a lot of attention in the years around 2010, as General Purpose Computing on GPU (GPGPU) became readily available to the scientific community. The beginning of the usage of GPUs for cryptanalytic purposes is marked by [34], including

elliptic curve cryptography, an early implementation of ECM on graphics cards was published in [4, 8]. A discussion of the performance of ECM on available GPUs around 2010 is given in [12, 32]. With the application of ECM in the cofactorization step of the GNFS, the discussion of an implementation for GPUs that includes ECM's second stage on the GPU was published in [31].

Existing Implementations. Although many authors have already worked on implementing ECM on GPUs, only a few implementations are openly available. **GMP-ECM**¹, which features support for computing the first stage of ECM on graphic cards using Montgomery curves, was introduced by Zimmermann et al.

Bernstein et al. published GPU-ECM in [8] and an improved version CUDA-ECM in [4]. In the following years, Bernstein et al. [5] released **GMP-EECM** – a variant of **GMP-ECM** using Edwards curves –, and subsequently **EECM-MPFQ**, which is available online at <https://eecm.cr.yp.to/>. Both latter, however, do not support GPUs.

To the authors' knowledge, the most recent implementation, including ECM's second stage by Miele et al. in [31] has not been made publicly available. Additionally, almost all previous implementations of ECM on GPUs only consider a fixed set of parameters for the bit length and ECM bounds. As we will show in Section 2, these restrictions do not meet real world assumptions and scalability seems to be significant even for larger moduli.

Contribution. We propose a complete and scalable implementation of ECM suitable for NVIDIA GPUs, that includes:

- 1. State-of-the-art Fast Curve Arithmetic** All elliptic curve computations are performed using $a = -1$ Twisted Edwards curves with the fastest known arithmetical operations.
- 2. Scalability to Arbitrary ECM Parameters** We show that currently used parameters for ECM in related work do not meet assumptions in realistic scenarios as most implementations are optimized for a set of small and fixed problem sizes. Hence, we propose an implementation which can be easily scaled to any arbitrary ECM parameter and bit length.
- 3. State-of-the-art Integer Arithmetic** We demonstrate that the commonly used CIOS implementation strategy can be outperformed by the less widespread FIOS and FIPS approaches on modern GPUs.
- 4. No Limitation to any Specific GPU Generation** Our implementation uses GPU-generation independent low level code based upon the PTX-level abstraction.

The corresponding software is released under an open source license and is available at <https://github.com/Chair-for-Security-Engineering/ecmongpu>.

Outline. The remainder of this paper is organized as follows: Sect. 2 briefly summarizes the DLP and the background of ECM. In Sect. 3 we describe our

¹ Available at <http://ecm.gforge.inria.fr/>.

optimizations for stage one and stage two on the algorithmic level. This is followed by Sect. 4 introducing our implementation strategies for GPUs. Before concluding this work in Sect. 6, we evaluate and compare our implementation in terms of throughput in Sect. 5.

2 Preliminaries

This section provides the mathematical background of ECM and introduces cofactorization as part of GNFS.

2.1 Elliptic Curve Method

ECM introduced by H. W. Lenstra in [28] is a general purpose factoring algorithm which works on random elliptic curves defined modulo the composite number to be factored. Thus, ECM operates in the group of points on the curve. Whether ECM is able to find a factor of the composite depends on the smoothness of the order of the curve. Choosing a different random curve most likely results in a different group order. To increase the probability of finding a factor, ECM can be run multiple times (in parallel) on the same composite number. ECM consists of a first stage and an optional second stage.

Stage 1. In the first stage, one chooses a random curve E over \mathbb{Z}_n with n the composite to factor, with p being one of its factors, and a random point P on the curve. With s a large scalar, one computes the scalar multiplication sP and hopes that $sP = \mathcal{O}$ (the identity element) on the curve modulo p , but not modulo n . As p is unknown, all computations are done on the curve defined over \mathbb{Z}_n .

This can be regarded as working on all curves defined over \mathbb{Z}_p for all factors p simultaneously. If p was known, reducing the coordinates of a point computed over \mathbb{Z}_n modulo p yields the point on the curve over \mathbb{Z}_p .

If s is a multiple of the group order, i.e., the number of points on the curve over \mathbb{F}_p , sP is equal to the point at infinity $\mathcal{O} = (0 : 1 : 0)$ modulo p , e.g., on Weierstrass curves, but not n . Thus, the x - and z -coordinates are a multiple of p , and so $\gcd(x, n)$ (or $\gcd(z, n)$) should reveal a factor of n .

One chooses s to be the product of all small powers of prime numbers p_i up to a specific bound B_1 , i.e., $s = \text{lcm}(1, 2, 3, \dots, B_1)$. If the number of points on the chosen curve $\#E$ modulo p divides s , a factor will be found by ECM. This is equivalent to stating that the factorization of $\#E$ consists only of primes $\leq B_1$, thus is B_1 -smooth.

Stage 2. Stage two of ECM relaxes the constraint that the group order on E has to be a product of primes smaller than B_1 and allows one additional prime factor larger than B_1 but smaller than a second bound B_2 .

Thus, for $Q = sP$ the result from stage one, for each prime p_i with $B_1 < p_1 < p_2 < \dots \leq B_2$, one computes $p_i Q = p_i sP$ and hopes that $p_i s$ is a multiple of the group order. If so, $p_i Q$ – as in stage one – is equivalent to the identity element modulo p , the x - and z -coordinates equal 0 modulo p but not modulo n , hence the gcd of the coordinates and n reveals a factor of n .

Curve Selection and Construction. As ECM's compute intensive part is essentially scalar multiplication, we chose $a = -1$ Twisted Edwards curves [6] with extended projective coordinates [19] as these offer the lowest operation costs for point additions and doublings. Each point $P = (X : Y : T : Z)$ is thus represented by four coordinates, each of the size of the number to factor.

As ECM works on arbitrarily selected elliptic curves modulo the number to factor, multiple parameterized curve constructions have been proposed (see [37] for an overview). Our implementation constructs random curves according to the proposal by G elin et al. in [17].

2.2 Discrete Logarithm Problem

In 2016 the DLP was solved for a 768-bit prime p [23]. The computation of a database containing discrete logarithms for small prime ideals took about 4000 CPU core years. Using this database, an individual discrete logarithm modulo p could be computed within about two core days. Using more than one CPU core, the latency could be decreased, but the parallelization is not trivial. Recently, Boudot et al. announced a new record, solving the DLP for a 795-bit prime [13].

The computation of an individual logarithm of a number z consists of two computationally intensive steps: the initial split and the descent. During the initial split the main task is to find two smooth integers that are norms of certain principal ideals, such that their quotient modulo p equals z . The prime factors of these two integers correspond to prime ideals with not too large norms. During the descent step, each of these prime ideals can be replaced by smaller ideals using relations found by sieving realized in the same way as during the sieving step in the first step of the GNFS. Eventually, all prime ideals are so small, that their discrete logarithms can all be found in the database. These discrete logarithms can easily be assembled to the discrete logarithm of the number z .

The initial split is done by first performing some sieving in some lattice. The dimension of this lattice can be two or eight for example, depending on the number fields. This produces a lot of pairs of integers. There are many lattices that can be used for sieving, which offer obvious opportunities for parallelization and lead to even more pairs of integers. It is enough to find just one pair such that both integers are smooth enough. Smoothness of integers can be tested by a combination of several factorization algorithms. The most popular are trial division, Pollard- $(p - 1)$ and ECM.

One goal of our work was to reduce the latency of two CPU core days for the computation of individual 786-bit discrete logarithms using 25 CPUs with 4 cores each (Intel Core i7-4790K CPU @ 4.00GHz). In the initial split it is important to find good parameters for the factorization algorithms. For our purpose we found that

$$\begin{aligned} B_1 &\approx 7 \cdot \exp(n/9) \\ B_2 &\approx 600 \cdot \exp(0,113 \cdot n) \end{aligned}$$

are good choices for ECM to detect an n -bit factor ($n \in \{44, 45, \dots, 80\}$) using our CPUs. This is close to the widely used $B_2 \approx 100 \cdot B_1$. The sieving of the

descent step was parallelized with Open MPI and the sieving strategy was carefully chosen and balanced with the factorization strategies used in the initial split. Finally, we managed to compute individual discrete logarithms on 25 CPUs (i.e., 100 cores) within three minutes.

The implementation of ECM on GPUs provides several opportunities to speed up the computation of discrete logarithms. First, it can be used for smoothness testing in the sieving step in the first step of the GNFS in order to reduce the 4000 core years by supporting the CPUs with GPUs. Second, in the same way it can speed up the sieving in the descent step. Third, it can be used for speeding up the smoothness tests in the initial split.

In our experiment we utilized two GeForce RTX 2080 TI GPUs filtering the pairs of integers in the initial split between the sieving and the smoothness tests. The parameters of the sieving in the initial split were relaxed, such that the sieving was faster, producing more pairs (and reducing their quality). This leaves us with a huge amount of pairs of integers, most of them not smooth enough. These integers were reduced to a size of 340 bit at most by trial division (or otherwise dropped). The surviving integer pairs were sent to the two GPUs in order to find factors with ECM using two curves, $B_1 = 5\,000$, and $B_2 = 20\,000$. A pair survived this step, if ECM found a factor in both integers and after division by this factor the integers were still composite. The remaining survivors were sent to the GPUs in order to find factors with ECM using 50 curves, $B_1 = 5\,000$, and $B_2 = 30\,000$. The final survivors were sent to a factorization engine on CPUs. Eventually, the use of GPUs reduced the latency of the computation of individual logarithms from three minutes to two minutes.

To this end, the aforementioned experiments demonstrate that our ECM implementation on GPUs can support the GNFS substantially, speeding up the computation of discrete logarithms and possibly also the factorization of RSA moduli with the GNFS.

After building a database for a prime p , individual discrete logarithms can be computed rather easily. We estimate the cost for building such a database within a year using CPUs to roughly 10^6 US\$ for 768 bit, to 10^9 US\$ for 1024 bit and to at least 10^{14} US\$ for 1536 bit. In our experiments we could compute individual logarithms for 1024 bit within an hour on 100 CPU cores (up to the point of looking up in a database which we did not have). This is an upper bound since we did not focus on optimizations on polynomial selection and on choosing good parameters and a good balance between initial split and descent. The initial split produced 448-bit integers after trial division and the parameters for ECM went up to $B_1 = 50\,000$ and $B_2 = 5\,000\,000$. Due to the opportunity to scale our ECM implementation to any arbitrary parameter set, these numbers can be processed on GPUs which should considerably reduce the latency of one hour for 1024-bit individual logarithms.

3 Algorithmic Optimizations

With the general algorithm and background of ECM discussed in Sect. 2.1, this section introduces optimizations to both stages of the algorithm suitable for efficient GPU implementations.

3.1 Stage 1 Optimizations

As introduced in Sect. 2.1, during stage one of ECM a random point P on an elliptic curve is multiplied by a large scalar $s = \text{lcm}(1, 2, \dots, B_1 - 1, B_1)$, e.g., for a $B_1 = 50\,000$ s is 72115 bit. To this end, stage one of ECM is essentially a scalar multiplication of a point on an elliptic curve. This section will deal with the possible optimizations, leading to a faster computation of $s \cdot P$ for large s . This section introduces methods for reducing that effort.

Non-Adjacent Form. In general, our implementation uses a w -NAF (Nonadjacent form) representation for the scalar $s = \sum_{i=0}^{t-1} s_i 2^i$, where $s_i \in C = \{-2^{w-1} + 1, -2^{w-1} + 3, \dots, -1, 0, 1, \dots, 2^{w-1} - 1\}$. While the necessary point doublings roughly stay the same, the number of point additions is reduced at the cost of needing to precompute and store a small multiple of the base point for each positive coefficient from C . For example, choosing $w = 4$ reduces the number of point additions to 14 455 for a $B_1 = 50\,000$ at the cost of storage for three additional points ($3P, 5P, 7P$).

The w -NAF representation of any scalar can be computed on-the-fly during program startup. For all upcoming experiments we decided to set $w = 4$ allowing a fair comparison and removing one degree of freedom.

Different Scalar Representations. For *fixed* values of B_1 used repeatedly, other representations of the scalar can be found with significantly more precomputation. Addition chains have been proposed by Bos et al. in [12], however finding (optimal) addition chains with low operation cost for large scalars is still an open question. In [15] Dixon et al. also proposed so-called batching for splitting the scalar s into batches of primes in order to lower the overall number of required point operations. In [9] Bernstein et al. introduced fast tripling formulas for points on Edwards curves and presented an algorithm finding the optimal chain for a target scalar s with the lowest amount of modular multiplications. Bouvier et al. also used tripling formulas and employed double-base chains and double-base expansions in combination with batches to generate multiple chains to compute scalars for somewhat larger bounds in [14]. Recently Yu et al. provided a more efficient algorithm to compute optimal double-base chains in [36].

However, these approaches are limited to small bounds B_1 (i.e., for $B_1 \leq 8\,192$) and therefore do not match our requirements of an arbitrary value for B_1 . Nevertheless, we generated double-base chains for small values of B_1 with the algorithm from [9] choosing $S = \pm\{1, 3, 5, 7\}$ and benchmarked them.

The two approaches – batching and addition chains – can be combined by generating multiple addition chains, one for each batch [12, 14]. We also included

Table 1. Comparison of different strategies optimizing the ECM throughput for stage one setting $B_1 = 8\,192$ and the modulus size to 192 bit. To count modular multiplications, we assume $1M = 1S$.

B_1	Optimal Chains [9]			4-NAF			Random Batching			Adapted from [14]		
	M^b	I^c	$\frac{\text{trials}}{\text{second}}$	M^b	I^c	$\frac{\text{trials}}{\text{second}}$	M^b	I^c	$\frac{\text{trials}}{\text{second}}$	M^b	I^c	$\frac{\text{Trials}}{\text{second}}$
4 096	48 442	4	311 032	49 777	4	354 422	48 307	20	294 466	N/A		
8 192	N/A ^a			99 328	4	215 495	95 756	64	163 751	90503	0	138565
50 000	N/A ^a			605 983	4	37 476	585 509	432	25 718	N/A		

^a The calculation of an optimal chain is too computation-intensive ^b Modular multiplications
^c Modular inversions (during computation of small multiples and/or point optimization)

results for a slightly modified version of the chains from [14]. We used their batching but generated *only* optimal double-base chains using the code from [9] with $S = \pm 1$ (no precomputation), whereas the authors use 22 double-base expansions and switch to Montgomery coordinates for 4 batches out of a total of 222 batches. As a result, our variant needs to perform 931 additional modular multiplications. We disabled our optimized point representation (see Sect. 4.2) due to the high number of chains resulting in many costly inversions.

While in general the best batching strategy for larger B_1 is unclear, we were able to generate multiple addition chains for a $B_1 = 50\,000$ by randomly selecting subsets of primes smaller than B_1 and using the algorithm from [9]. Keeping only the best generated chains, we continued generating new batching variants for the rest of the primes still to cover until the overall cost of the chains stabilized. This strategy will be called *Random Batching* in the following. We supply all generated batched double-base addition chain for our B_1 with the software.

Table 1 compares the ECM stage one’s throughputs for $B_1 \in \{4\,096, 8\,192, 50\,000\}$ using the naive 4-NAF approach, our random batching, the results from [9] and our adaptation of [14] on an NVIDIA RTX 2080Ti. Although the batching based approaches require less modular operations (also compared to an optimal chain for $B_1 = 4\,096$), the absolute throughput is drastically lowered. We found that in practice the cost of using multiple chains quickly remedied the benefit of requiring less point operations: For each chain one needs to compute small multiples of the (new) base point when using a larger window size. Our implementation stores precomputed points in a variant of affine coordinates to reduce the cost of this point’s addition, each requiring one inversion during precomputation (cf. Sect. 4.2). This approach is not well suited if precomputed points are only used for relatively few additions on a single chain.

In addition, for each digit in *double-base* chains the software has to check whether it has to perform a doubling or tripling. Even if using only the base point and disabling the optimization of its coordinates, the overhead introduced by the interruption of the GPU program flow between chains slows down the computation, even though the full set of batches are processed on the GPU with one kernel launch.

In our experiments, we found that using our optimized coordinates for point addition with w -NAF scalar representation is more beneficial to the overall throughput than using multiple addition chains without the optimization of pre-computed points. Hence, our NAF approach achieves better results as only doublings are executed, the program flow is uninterrupted and no switching between operations is needed.

3.2 ECM Stage 2 Optimizations

As introduced above, in the second stage of ECM one hopes that the number of points on E is B_1 -powersmooth, except for one additional prime factor. For stage two, a second bound B_2 is set, and each multiple of the result of stage one $p_{k+i}Q$ for each prime $B_1 < p_{k+1} < p_{k+2} < \dots < p_{k+l} \leq B_2$ is computed.

Reducing Point Operations. The number of point operations can be reduced by employing a baby-step giant-step approach as in [30]. Each prime p_{k+i} is written as $p_{k+i} = vg \pm u$, with g a giant-step size and u the number of baby-steps. To cover all the primes between B_1 and B_2 , set

$$u \in U = \left\{ u \in \mathbb{Z} \mid 1 \leq u \leq \frac{g}{2}, \gcd(u, g) = 1 \right\}$$

$$v \in V = \left\{ v \in \mathbb{Z} \mid \left\lceil \frac{B_1}{g} - \frac{1}{2} \right\rceil \leq v \leq \left\lfloor \frac{B_2}{g} + \frac{1}{2} \right\rfloor \right\}.$$

As in stage two, one tries to find a prime $p_{k+i} = vg \pm u$ such that $(vg \pm u)Q = \mathcal{O}$ on the curve modulo a factor p . This is equivalent to finding a pair of vg and u , such that $vgQ = \pm uQ \pmod p$. If this is the case, then the (affine) y -coordinates of vgQ and uQ are also equal and

$$\frac{y_{vgQ}}{z_{vgQ}} - \frac{y_{uQ}}{z_{uQ}} = 0 \pmod p.$$

Since $\frac{y_P}{z_P} = \frac{y(-P)}{z(-P)}$ on Twisted Edwards curves, one only needs to check for $y_{vgQ}z_{uQ} - y_{uQ}z_{vgQ}$, if either $vg + u$ or $vg - u$ is a prime, thus saving computation on roughly half the prime numbers. Our implementation uses a bitfield to mark those combinations that are prime.

The result of the difference for all l primes of y -coordinates can be collected, so that stage two only outputs a single value m with

$$m = \prod_{v \in V} \prod_{u \in U} y_{vgQ}z_{uQ} - y_{uQ}z_{vgQ}.$$

If any of the differences $y_{vgQ}z_{uQ} - y_{uQ}z_{vgQ}$ equals zero modulo p , $\gcd(m, n)$ is divisible by p and usually not n thus a non-trivial factor of n is found.

When for all $u \in U$ points the point uQ is precomputed together with the giant-step stride of gQ , this approach only needs $|V| + |U| + 1$ point additions, plus $3|V||U|$ modular multiplications for the computation of m .

Reducing Multiplications. Our approach is to normalize all points vgQ and uQ to the *same* projective z -coordinates instead of affine coordinates. This way the computation of m only requires y -coordinates, because – as introduced above – the goal is to find equal points modulo p . Given $a \geq 2$ points P_1, \dots, P_a – in this case all giant-step points vgQ and baby-step points uQ – the following approach sets all z_{P_i} to $\prod_{1 \leq i \leq a} z_{P_i}$: To do so, each z_{P_i} needs to be multiplied by $\prod_{1 \leq i \leq a, i \neq k} z_{P_i}$. An efficient method to compute each $\prod_{1 \leq i \leq a, i \neq k} z_{P_i}$ is given in [24, p. 31].

Normalizing all points to the same z -coordinate costs $4(|V| + |U|)$ multiplications during precomputation and the cost of computing m drops down to $|V||U|$ modular multiplications, as $m = \prod_{v \in V} \prod_{u \in U} y_{vgQ} - y_{uQ}$.

However, for this normalization all baby- and giant-step points need to be precomputed which needs quite a lot of memory to store z - and y -coordinates of all $|V| + |U|$ baby-step and giant-step points, as well as the storage of the batch cross multiplication algorithm from [24, p. 31] with $|V||U|$ entries. If less memory is available, the giant-step points can be processed in batches. In this case, the normalization has to be computed again for each batch.

4 Implementation Strategies

The following sections discuss in more detail the implementation of multi-precision arithmetic and elliptic curve operations tuned to our requirements and those of GPUs.

4.1 Large Integer Representation on GPUs

Our implementation follows the straight-forward approach of, e.g., [29, 31] and uses 32-bit integer limbs to store multi-precision values. The number of limbs for any number is set at compile time to the size of the largest number to factor. Thus, all operations iterating over limbs of multi-precision numbers, can be completely unrolled during compilation, yielding long sequences of straight-line machine code. To avoid inversions during multi-precision arithmetic, all computation on the GPU is carried out in the Montgomery domain. All multi-precision arithmetic routines use inline Parallel Thread Execution (PTX) assembly to make use of carry-flags and multiply-and-add instructions. Note that PTX code, while having an assembly-like syntax, is code for a virtual machine that is compiled to the actual architecture specific instructions set. PTX has the advantage of being hardware independent and ensures our proposed implementation is executable on a variety of NVIDIA hardware.

To enable fast parallel transfer of multi-precision values from global device memory to registers and shared memory of the GPU cores, multi-precision values in global memory are stored strided: Consecutive 32-bit integers in memory are arranged such that they are retrieved by different GPU threads, thus coalescing memory accesses to the same limb of different values by multiple threads into one memory transaction.

GPU-Optimized Montgomery Multiplication. As the modular multiplication is at the core of elliptic curve point operations, the speed of the implementation is most influenced by the speed of the modular multiplication routine. As in the implemented software architecture, a single thread performs the full multiplication to avoid any synchronization overhead between threads, reducing the amount of registers per multiplication is of high importance.

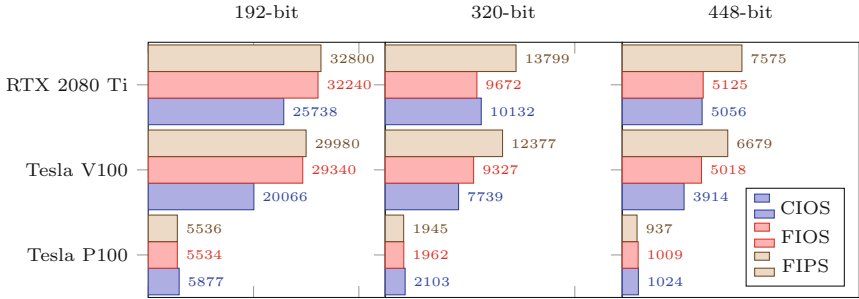


Fig. 1. Million modular multiplication per second for different Montgomery implementation strategies and architectures.

Different strategies to implement multi-precision Montgomery multiplication and reduction have been surveyed in [20]. These differ in two aspects: The tightness of interleaving of multiplication and reduction, and the access to operands' limbs. In [32], Neves et al. claimed that the Coarsely Integrated Operand Scanning (CIOS), Finely Integrated Operand Scanning (FIOS) and Finely Integrated Product Scanning (FIPS) strategies are the most promising, and CIOS is most widely used, e.g., in [34]. All three methods need $2l^2 + l$ multiplications of limbs for l -limb operands (see [20, Table 1] for a complete cost overview). Using PTX, each of these multiplications requires two instructions to retrieve the lower and upper half of the $2l$ product. PTX offers multiply-and-add instructions with carry-in and -out to almost entirely eliminate additional `add` instructions.

Our implementation of FIOS follows [18] in accumulating carries in an additional register to prevent excessive memory accesses and carry propagation loops. Our FIPS implementation follows [32, Algorithm 4].

Comparing FIPS, FIOS and CIOS on current GPUs, our benchmarks show varying results for newer architectures. Figure 1 shows the runtime of different strategies on different hardware architectures. For each of these benchmarks, 32 768 threads are started in 256 blocks, with 128 threads in each block. Each thread reads its input data from strided arrays in global memory and performs one million multiplications (reusing the result as operand for the next iteration) and writes the final product in strided form back to global memory.

For the most recent Volta and Turing architectures featuring integer arithmetic units, the FIPS strategy is the most efficient especially for larger moduli. On the older Pascal architecture, the difference between the implementation

strategies' efficiency is much smaller. However, on the Tesla P100 CIOS slightly outperformed both finely integrated methods.

GPU-Optimized Montgomery Inversion. While modular inversions are costly compared to multiplications and are not used during any *hot* arithmetic, pre-computed points are transformed needing one modular inversion per point. Montgomery Inversion, given a modulus n and a number $\tilde{A} = AR$ to invert in Montgomery representation, computes its inverse $\tilde{A}^{-1} = A^{-1}R \pmod n$, again in Montgomery representation.

The algorithm implemented in this work is based on the Binary Extended Euclidean Algorithm as in [33, Algorithm 3]. Divisions by two within the algorithm are accomplished by using PTX *funnel shifts* to the right. The PTX instruction `shf.r.clamp` takes two 32-bit numbers, concatenates them and shifts the 64-bit value to the right, returning the lower 32 bit. Thus, each division by two can be achieved with l instructions for an l -word number. However, the inversion algorithm needs four branches depending on the number to invert and thus produces inner warp thread divergence.

4.2 Elliptic Curve Arithmetic on GPUs

Based on the modular arithmetic of the last section, the elliptic curve arithmetic can be implemented. With offering the lowest operation count (in terms of multiplications/squarings) of all proposed elliptic curves, our GPU implementation uses $a = -1$ twisted Edwards curves, with coordinates represented in extended projective format.

Point Arithmetic. The implementation of point addition and subtraction is a straight-forward application of the addition and doubling formulas from [19] using the multi-precision arithmetic detailed in the previous section.

Point Addition. Addition of an arbitrary point with $Z \neq 1$ is only needed seldom: During precomputation of small multiples of the base point for the w -NAF multiplication and during computation of the giant-steps for stage two. General point addition is implemented by a straight-forward application of the formulas from [10, 19] as given in Algorithm 1.

Algorithm 1: Point addition on $a = -1$ twisted Edwards curves [10, 19].

Data: Points $P = (x_P, y_P, z_P, t_P)$ and $Q = (x_Q, y_Q, z_Q, t_Q)$ in extended projective coordinates, curve parameter $k = 2d$		
Result: Point $R = P + Q = (x_R, y_R, z_R, t_R)$		
1 $a \leftarrow (y_P - x_P) \cdot (y_Q - x_Q)$	6 $e \leftarrow b - a$	11 $y_R \leftarrow g \cdot h$
2 $b \leftarrow (y_P + x_P) \cdot (y_Q + x_Q)$	7 $f \leftarrow d - c$	12 $z_R \leftarrow f \cdot g$
3 $c \leftarrow t_P \cdot k \cdot t_Q$	8 $g \leftarrow d + c$	13 $t_R \leftarrow e \cdot h$
4 $d \leftarrow z_P \cdot z_Q$	9 $h \leftarrow b + a$	14 return
5 $d \leftarrow d + d$	10 $x_R \leftarrow e \cdot f$	(x_R, y_R, z_R, t_R)

Table 2. Modular operation cost of the implemented point arithmetic.

	projective*			extended*		
	M	S	ADD	M	S	ADD
Doubling [†]	3	4	8	4	4	8
Tripling [†]	9	3	10	11	3	10
Addition*	8		9	9		9
Precomputed addition [‡]	6		7	7		7

* result coordinate format † operand in projective coordinates * operand in extended coordinates ‡ one operand in our modified coordinates

If one of the points of the addition is precomputed and used in many additions, further optimization is beneficial. As in the w -NAF point multiplication, precomputed points are only used for addition, all operations that solely depend on values of the point itself are done once during precomputation. These are addition and subtraction of x - and y -coordinates, as well as the multiplication of the t -coordinate with the curve constant $k = 2d$. To further save one multiplication per point addition, the precomputed point can be normalized such that its z -coordinate equals one at the cost of one inversion and three multiplications. Applying these optimizations yields the modified format of a precomputed point \tilde{P} from the general point representation P , such that

$$x_{\tilde{P}} = y_P - x_P \quad y_{\tilde{P}} = y_P + x_P \quad z_{\tilde{P}} = 1 \quad t_{\tilde{P}} = 2 \cdot d_{curve} \cdot t_P$$

Using this representation, point additions require seven multiplications only. Computing the inverse of a point $-P = (-x_P, y_P, z_P, -t_P)$ in its modified representation is achieved by switching the x - and y -coordinates, and computing $-t_{\tilde{P}} = n - t_{\tilde{P}} \pmod n$, i.e., $-\tilde{P} = (y_{\tilde{P}}, x_{\tilde{P}}, 1, n - t_{\tilde{P}})$.

Point Doubling and Tripling. Point doubling is used for each digit of the scalar in scalar multiplication, tripling also on double-base chains. As all intermediate values do not fulfill the condition of $Z = 1$, no further optimized doubling formulas can be applied in this case. The implemented doubling and tripling routines follow [10, 19] and [9].

Mixed Representation. Using extended projective coordinates, the point doubling formula does not use the t -coordinate of the input point. When using the w -NAF scalar multiplication, the number of non-zero digits is approximately $\frac{l}{w-1}$ for an l -bit scalar. Thus, there are long runs of zero bits in the w -NAF, resulting in many successive doublings without intermediate addition.

Thus, to further reduce multiplications during scalar multiplication computing the t -coordinate can be omitted if the scalar's next digit is zero, as no addition follows in this case. Furthermore, as each point addition is followed by a point doubling, which does not rely on the correct extended coordinate, again,

the multiplication computing t_R can be omitted from all point additions within the scalar multiplication. The same applies to tripling. The resulting operation counts as implemented are listed in Table 2.

Scalar Multiplication. To compute the scalar multiple of any point P , as in the first stage of ECM, w -NAF multiplication is used. The first stage's scalar $s = \text{lcm}(1, \dots, B_1)$ is computed on the host and transformed into w -NAF representation, with w a configurable compile time constant defaulting to $w = 4$. Thus, each digit of $s_{w\text{-NAF}}$ is odd or zero and in the range of -2^{w-1} to 2^{w-1} .

Our precomputation generates $2P$ by point doubling and the small odd multiples of P , i.e., $\{3P, \dots, (2^{w-1} - 1)P\}$ with repeated addition of $2P$. Precomputed points are stored with strided coordinates along with other batch data in global memory, as registers and shared memory are not sufficiently available.

All threads read their corresponding precomputed point's coordinates from global memory to registers with coalesced memory accesses. In case the current digit of the NAF is negative, the precomputed point is inverted before addition. Again, as all threads are working on the same limb, this does not create any divergence.

5 Evaluation

Three different GPU platforms were available during this work, a Tesla P100 belonging to the Pascal family, a Tesla V100 manufactured in the Volta architecture, and a RTX 2080 Ti with a Turing architecture.

As the actual curves in use for ECM are not within the scope of this paper, the *yield*, i.e., the numbers for which a factor is found, is not part of this evaluation. Of interest is, however, the throughput of the implementation: *How many ECM trials can be performed per second on moduli of a given bit length*. Therefore, each benchmark in this work is conducted on 32 768 randomly generated numbers $n = pq$, with $\sqrt{n} \approx p \approx q$ and p and q prime.

Benchmarks for different problem sizes are carried out in two standard configurations, with the first being a somewhat standard throughout the literature to enable a comparison with previous works. As most previously reported GPU implementations only support the first stage of ECM on the GPU, this first case only executes stage one of the implementation with a bound of $B_1 = 8\,192$. The second benchmark parameter set is aimed at much larger ECM bounds and does include the second stage, with bounds $B_1 = 50\,000$ and $B_2 = 5\,000\,000$.

5.1 Stage One Bound

Firstly, we evaluate the impact of the bound B_1 . Figure 2 gives the number of ECM trials per second for moduli of 192 bit and 320 bit for growing values of B_1 . Note that the size of the scalar $s = \text{lcm}(1, \dots, B_1)$ grows very fast with B_1 . Using w -NAF multiplication, the runtime of ECM mainly depends on the number of digits in s , resulting in the values seen in Fig. 2. Note that each single trial (per

second) on the y -axis is equivalent to $\log_2 \text{lcm}(1, \dots, B_1)$ operations (double and possibly add) per second and thus changes for each value of B_1 , e.g., 1 trial is equivalent to 14 447 ops for $B_1 = 10\,000$ and 28 821 ops for $B_1 = 20\,000$.

5.2 Stage Two Bound

For a given bound B_2 , the number of primes less than or equal to B_2 are the key factor in determining the runtime of stage two. Via the prime number theorem, with a fixed negligible value for B_1 , this value is approximately $\pi(B_2) \approx \frac{B_2}{\ln B_2}$. See Fig. 3 for the achieved ECM trials per second for different values of B_2 . While for small values of B_2 , the RTX 2080 Ti outperforms the Tesla V100, as soon as B_2 grows larger than 1 000 000, the Tesla V100 performs slightly better. As described in Sect. 3.2 for larger values of B_2 not all baby-step and giant-step points can fit into GPU memory, but have to be processed in batches. Our Tesla V100 setup features 16 GB of GPU memory while the RTX 2080 Ti only has 11 GB available. Again, note that the plot shows $\frac{\text{trials}}{\text{second}}$ where with growing B_2 the number of operations per trial increases with B_2 .

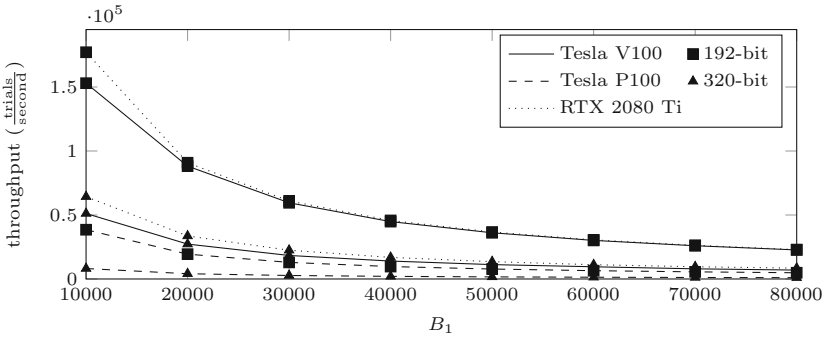


Fig. 2. ECM first stage trials per second for varying size of B_1 .

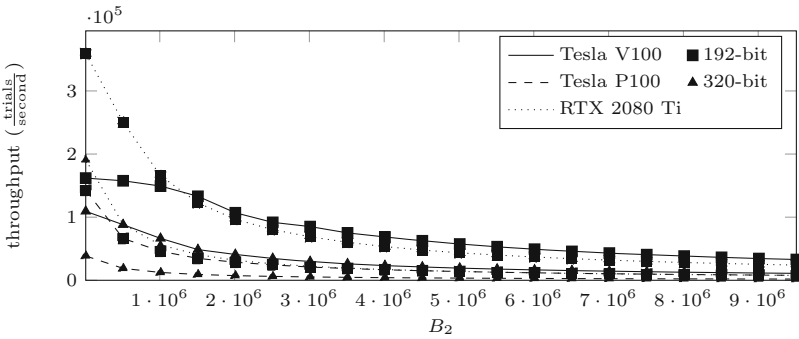


Fig. 3. ECM first and second stage trials per second for varying size of B_2 , with $B_1 = 256$, and a stage two window size of $w = 2310$ (cf. Sect. 3.2).

5.3 ECM Throughput

With these benchmarks giving the runtime dependency on different parameters, this section gives absolute throughput numbers for the two exemplary cases of first stage only ECM with $B_1 = 8192$, and both stages with more ambitious $B_1 = 50\,000$ and $B_2 = 5\,000\,000$.

Stage 1. The absolute throughput for the first case for different moduli sizes is given in Table 3. Interestingly, when comparing the throughput for 192-bit moduli between the high-performance GPU Tesla V100 with the consumer GPU RTX 2080 Ti, the consumer card processes more ECM trials per second by a factor of 1.44.

Table 3. Absolute throughput of ECM trials for stage one (in thousands per second) on different platforms with $B_1 = 8192$ and varying moduli sizes.

	128	160	192	224	256	288	320	352	384	416	448
Tesla P100	103.9	66.6	46.8	33.5	19.0	14.3	9.9	8.3	7.0	6.0	5.2
Tesla V100	228.9	188.8	149.1	141.3	117.6	73.4	61.9	52.4	35.4	29.4	24.7
RTX 2080 Ti	450.6	310.0	214.1	152.5	124.2	98.9	77.1	58.8	37.2	29.7	24.7
2×RTX 2080 Ti	542.6	481.3	377.1	285.5	232.9	191.4	150.2	113.6	73.0	58.3	48.2

Table 4. Absolute throughput of ECM trials for stage one and stage two (in thousands per seconds) on different platforms with $B_1 = 50\,000$, $B_2 = 5\,000\,000$ and varying moduli sizes.

	128	160	192	224	256	288	320	352	384	416	448
Tesla P100	10.79	7.15	4.97	3.52	1.91	1.42	1.11	0.91	0.77	0.65	0.55
Tesla V100	46.88	30.74	22.85	17.12	13.58	7.99	7.12	5.78	4.60	3.49	2.78
RTX 2080 Ti	40.86	27.39	20.21	14.77	11.62	9.34	6.85	6.30	4.11	3.32	2.78
2×RTX 2080 Ti	80.46	53.79	39.42	28.61	22.51	17.91	13.50	9.72	7.89	6.46	5.39

Stage 1 and Stage 2 Eventually, Table 4 states the absolute throughput of the entire ECM setting the bounds to $B_1 = 50\,000$ and $B_2 = 5\,000\,000$. For the exemplary application with a modulus size of 448 bit mentioned in Sect. 2.2, only one RTX 2080 Ti is capable of processing 2781 ECM trials per second.

Multiple Devices Our implementation is designed to use multiple GPUs to increase throughput. Table 3 and Table 4 show that the throughput is almost doubled when utilizing two RTX 2080 Ti, and more so for larger moduli and larger ECM parameters, as the ratio of host side to GPU computation shifts towards more work on the GPU.

Table 5. Comparison of scaled throughput for Montgomery multiplication from the literature and this work. Throughput values are given in $\frac{\text{multiplications}}{\text{core} \times \text{cycle}} \times 10^{-3}$.

GPU	[25]	[31]	[16] ^d	this work		
	GTX 480	GTX 580	GTX 980 Ti ^a	Tesla P100 ^b	Tesla V100 ^c	RTX 2080 Ti ^c
Cores	480	512	2816	3584	5120	4352
Clock*	1401	1544	1000	1316	1530	1665
Modulus [†]						
128	3.54063	7.34319	4.03125	2.65388	9.01974	8.65915
160	2.85956	4.75631		1.74424	6.40737	6.01596
192	2.32423	3.32816		1.24673	4.65732	4.62363
224	1.90638	2.45785		0.91325	3.61875	3.46953
256	1.53313	1.88861	1.32813	0.70659	2.92659	2.80919
320	1.04687	1.21691		0.44531	1.97959	1.88013
384	0.75839	0.84880	0.64063	0.30303	1.41461	1.36107

* in MHz † in bits ^a two-pass approach ^b CIOS ^c FIPS

^d Values have been scaled from throughput per *Streaming Multiprocessor* per clock cycle

5.4 Comparison to Previous Work

Multiple factors make it hard to compare our results to previous work: Especially the fast changing GPU architectures make a comparison very difficult, but also no comparable set of parameters for B_1 and B_2 has been established. In lack of a better computation power estimate, we adopt the approach of [31] to scale the results accomplished on different GPU hardware by scoring results per cuda cores \times clock rate.

Montgomery Multiplication. Comparing the most influential building block, the Montgomery multiplication algorithm to previous publications is a first step. Table 5 lists relevant work, the hardware in use and a score for the throughput scaled by the number of Compute Unified Device Architecture (CUDA) cores and their clock rate. The implementation of this work is the fastest of all implementations under comparison on the RTX 2080 Ti and more so for larger moduli, however comes in last place for the Pascal architecture platforms. Using our implementation and a modern GPU manufactured in the Turing architecture, clearly outperforms the previous results.

ECM Throughput. Comparing the achieved throughput of the developed software with previously published results suffers from various problems: different hardware, varying modulus sizes and varying settings for both first and second stage bounds across different publications.

Especially, as to the authors' knowledge, apart from Miele et al. in [31], no other publication of ECM on GPUs implemented the second stage. Additionally, in [31] only very small bounds of $B_1 = 256$ and $B_2 = 16\,384$ were chosen. Note that the implemented w -NAF approach in stage one in this work benefits from larger B_1 as precomputation costs amortize.

Table 6. Comparison of this implementation with [12] and their parameter sets for 192-bit moduli. Values are given in $\frac{\text{ECM trials}}{\text{core} \times \text{cycle}} \times 10^{-5}$.

GPU	Bos et al. [12]		this work		
	no-storage	windowing			
cores/clock*	GTX 580		Tesla P100	Tesla V100	RTX 2080 Ti
	512/1544		3584/1316	5120/1530	4352/1665
$B_1 = 960$	2.1692	1.0014	0.64070	0.20936	0.49398
$B_1 = 8192$	0.2513	0.1151	0.09917	0.20134	0.29373
$B_1 = 50000$	N/A	N/A	.01650	0.04609	0.05168

* in MHz

For bounds this small our implementation is actually significantly slower, as host-side and precomputation overhead dominate the runtime.

Albeit already published in 2012, the comparison with [12] is the most interesting for the stage one implementation, as they also use a somewhat larger bound of $B_1 = 8192$, but do not implement stage two. However, the comparison lacks modulus sizes other than 192 bit, as [12] only published these results. The comparison to our implementation is shown in Table 6 and perfectly shows the advantage of our approach for larger bounds. Considering $B_1 = 8192$, our implementation slightly outperforms the *no-storage* approach by Bos et al. although we do not use highly optimized addition chains.

Even less recent, published in 2009, is the implementation by Bernstein et al. [8]. A comparison is somewhat unfair, as Bernstein developed $a = -1$ Edwards curves after this paper was published. However, their GPU implementation uses the bound $B_1 = 8192$, and in comparison the proposed implementation is significantly faster. However, this comparison is unfair as multiple generations of hardware architectures aimed at GPGPU have been released within the last ten years, and the authors of [8] decided to use a floating point representation.

6 Conclusion

In this work we present a highly optimized and scalable implementation of the entire ECM algorithm for modern GPUs. On algorithmic level, we demonstrated that a w -NAF representation seems to be the most promising optimization technique realizing the scalar multiplication in the first stage. For the second stage we rely on an optimized baby-step giant-step approach. For the underlying Montgomery multiplication, we implemented three difference strategies where against our expectations FIPS performs best. Eventually, we demonstrate that the throughput of previous literature is achieved – and actually exceeded – on the most recent Turing architecture. We hope that the scalability, flexibility and free availability of our ECM implementation will support other researchers in achieving new factorization and DL records, reducing costs and reassessing the security of some algorithms used in PKC.

References

1. Antao, S., Bajard, J.C., Sousa, L.: RNS-based elliptic curve point multiplication for massive parallel architectures. *Comput. J.* **55**(5), 629–647 (2012)
2. Antao, S., Bajard, J.C., Sousa, L.: Elliptic curve point multiplication on GPUs. In: ASAP 2010–21st IEEE International Conference on Application-specific Systems, Architectures and Processors. IEEE, July 2010
3. Barker, E.B., Dang, Q.H.: Recommendation for Key Management Part 3: Application-Specific Key Management Guidance. Technical Report NIST SP 800–57Pt3r1, National Institute of Standards and Technology, January 2015
4. Bernstein, D.J., et al.: The billion-mulmod-per-second PC. In: SHARCS 2009 Workshop Record (Proceedings 4th Workshop on Special-purpose Hardware for Attacking Cryptographic Systems, Lausanne, Switzerland, September 9–10, 2009) (2009)
5. Bernstein, D., Birkner, P., Lange, T., Peters, C.: ECM using edwards curves. *Math. Comput.* **82**(282), 1139–1179 (2013)
6. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted edwards curves. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 389–405. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68164-9_26
7. Bernstein, D.J., Birkner, P., Lange, T.: Starfish on strike. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 61–80. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14712-8_4
8. Bernstein, D.J., Chen, T.-R., Cheng, C.-M., Lange, T., Yang, B.-Y.: ECM on graphics cards. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 483–501. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_28
9. Bernstein, D.J., Chuengsatiansup, C., Lange, T.: Double-base scalar multiplication revisited. *Cryptology ePrint Archive*, Report 2017/037 (2017). <https://eprint.iacr.org/2017/037>
10. Bernstein, D.J., Lange, T.: Explicit-Formulas Database. <https://hyperelliptic.org/EFD/index.html>
11. Bos, J.W.: Low-latency elliptic curve scalar multiplication. *Int. J. Parallel Prog.* **40**(5), 532–550 (2012)
12. Bos, J.W., Kleinjung, T.: ECM at work. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 467–484. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_29
13. Boudot, F., Gaudry, P., Guillevic, A., Heninger, N., Thomé, E., Zimmermann, P.: Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment. *Cryptology ePrint Archive*, Report 2020/697 (2020). <https://eprint.iacr.org/2020/697>
14. Bouvier, C., Imbert, L.: Faster cofactorization with ECM using mixed representations. *Cryptology ePrint Archive*, Report 2018/669 (2018). <https://eprint.iacr.org/2018/669>
15. Dixon, B., Lenstra, A.K.: Massively parallel elliptic curve factoring. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 183–193. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-47555-9_16
16. Emmart, N., Luitjens, J., Weems, C., Woolley, C.: Optimizing Modular Multiplication for NVIDIA’s Maxwell GPUs. In: 2016 IEEE 23rd Symposium on Computer Arithmetic (ARITH), pp. 47–54. IEEE, Silicon Valley, CA, USA, July 2016

17. G elin, A., Kleinjung, T., Lenstra, A.K.: Parametrizations for families of ecm-friendly curves. In: Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2017, Kaiserslautern, Germany, July 25–28, 2017, pp. 165–171 (2017)
18. Gro sch adl, J., Kamendje, G.-A.: Optimized RISC architecture for multiple-precision modular arithmetic. In: Hutter, D., M uller, G., Stephan, W., Ullmann, M. (eds.) Security in Pervasive Computing. LNCS, vol. 2802, pp. 253–270. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-39881-3_22
19. Hisil, H., Wong, K.K.-H., Carter, G., Dawson, E.: Twisted edwards curves revisited. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 326–343. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_20
20. Kaya Koc, C., Acar, T., Kaliski, B.: Analyzing and comparing Montgomery multiplication algorithms. IEEE Micro **16**(3), 26–33 (1996)
21. Kleinjung, T., et al.: Factorization of a 768-Bit RSA modulus. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 333–350. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_18
22. Kleinjung, T., et al.: A heterogeneous computing environment to solve the 768-bit RSA challenge. Cluster Comput. **15**(1), 53–68 (2012)
23. Kleinjung, T., Diem, C., Lenstra, A.K., Priplata, C., Stahlke, C.: Computation of a 768-Bit prime field discrete logarithm. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 185–201. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_7
24. Kruppa, A.: A Software Implementation of ECM for NFS. Research Report RR-7041, INRIA (2009). <https://hal.inria.fr/inria-00419094>
25. Leboeuf, K., Muscedere, R., Ahmadi, M.: A GPU implementation of the Montgomery multiplication algorithm for elliptic curve cryptography. In: 2013 IEEE International Symposium on Circuits and Systems (ISCAS2013), pp. 2593–2596, May 2013
26. Lenstra, A.K.: Integer factoring. Des. Codes Crypt. **19**(2–3), 101–128 (2000). <https://doi.org/10.1023/A:1008397921377>
27. Lenstra, A.K.: General purpose integer factoring. Cryptology ePrint Archive, Report 2017/1087 (2017). <https://eprint.iacr.org/2017/1087>
28. Lenstra, H.W.: Factoring integers with elliptic curves. Ann. Math. **126**(3), 649–673 (1987). <https://doi.org/10.2307/1971363>
29. Mah e, E.M., Chauvet, J.M.: Fast GPGPU-based elliptic curve scalar multiplication. Cryptology ePrint Archive, Report 2014/198 (2014). <https://eprint.iacr.org/2014/198>
30. Miele, A.: On the analysis of public-key cryptologic algorithms (2015). <https://infoscience.epfl.ch/record/207710>
31. Miele, A., Bos, J.W., Kleinjung, T., Lenstra, A.K.: Cofactorization on graphics processing units. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 335–352. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_19
32. Neves, S., Araujo, F.: On the performance of GPU public-key cryptography. In: ASAP 2011–22nd IEEE International Conference on Application-specific Systems, Architectures and Processors, pp. 133–140. September 2011
33. Savas, E., Koc, C.K.: Montgomery inversion. J. Cryptographic Eng. **8**(3), 201–210 (2018)
34. Szerwinski, R., G uneysu, T.: Exploiting the power of GPUs for asymmetric cryptography. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 79–99. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_6

35. Valenta, L., Cohnsey, S., Liao, A., Fried, J., Bodduluri, S., Heninger, N.: Factoring as a service. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 321–338. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_19
36. Yu, W., Musa, S.A., Li, B.: Double-base chains for scalar multiplications on elliptic curves. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 538–565. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_18
37. Zimmermann, P., Dodson, B.: 20 years of ECM. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 525–542. Springer, Heidelberg (2006). https://doi.org/10.1007/11792086_37

Payment Systems



Arcula: A Secure Hierarchical Deterministic Wallet for Multi-asset Blockchains

Adriano Di Luzio, Danilo Francati^(✉), and Giuseppe Ateniese

Stevens Institute of Technology, Hoboken, NJ, USA
dfrancat@stevens.edu

Abstract. This work presents Arcula, a new design for hierarchical deterministic wallets that brings identity-based public keys to the blockchain. Arcula is built on top of provably secure cryptographic primitives. It generates all its cryptographic secrets from a user-provided seed and enables the derivation of new public keys based on the identities of users, without requiring any secret information. Unlike other wallets, it achieves all these properties while being secure against privilege escalation. We formalize the security model of hierarchical deterministic wallets and prove that an attacker compromising an arbitrary number of users within an Arcula wallet cannot escalate his privileges and compromise users higher in the access hierarchy. Our design works out-of-the-box with any blockchain that enables the verification of signatures on arbitrary messages. We evaluate its usage in a real-world scenario on the Bitcoin Cash network.

Keywords: Hierarchical deterministic wallet · Hierarchical key assignment · Bitcoin · Blockchain

1 Introduction

In recent years, the widespread adoption of distributed financial systems attracted the interest of millions of users, who have exchanged assets and coins in a decentralized and democratic way. Blockchain-based systems like Bitcoin, Ethereum, and Ripple achieve these goals—their users exchange coins by relying on distributed consensus, cryptographic keys, and without depending on any central authority. At the same time, however, the blockchain also attracted the interests of malicious parties that aimed at either compromising the distributed consensus or the cryptographic keys of users. In both cases, their goal is to steal crypto-coins.

This work focuses on this issue and, in particular, on the cryptographic wallets in which users store the cryptographic keys associated with their coins. Similarly to how we store bills and credit cards in our physical wallet, a cryptographic wallet contains the digital keys that allow us to spend crypto-coins on the blockchain (*e.g.*, the Bitcoins that we receive on an address we control).

A good wallet should be secure to external threats – *i.e.*, a physical or remote attacker should not be able to extract keys without the users’ consent – and also resilient to failures and disasters – the users should be able to recover their keys even in case of loss, hardware failure, or natural disaster. We focus, in particular, on a family of cryptographic wallets called hierarchical deterministic wallets (HDWs) that achieve these goals while providing innovative properties. For instance, HDWs allow users to distribute their cryptographic keys according to a hierarchy (reflecting, for example, their different blockchain addresses or the internal organization of a company). In addition, they deterministically generate all their secrets by starting from a user-provided *seed*. As long as users produce the seed, they will be able to recover all their keys. Finally, HDWs allow auditors (external to the wallet and unaware of any of its secrets) to verify the hierarchical relationships between the users and the corresponding keys that they control within the wallet. As we shall see, this last property enables an entirely new set of use cases on the blockchain and in the context of decentralized finance (DeFi).

Currently, HDWs are widely used both by individuals and financial enterprises. Nonetheless, many implementations of HDW are either broken from a security perspective, do not respect the fundamental properties of an HDW, or are exclusively tied to specific blockchains. In this work, we aim at solving this problem. We present Arcula,¹ a new design of HDW that is significantly different from the state of the art. First, Arcula is independent of any particular blockchain: It supports any crypto-system that enables the verification of signatures on arbitrary messages, and we test its implementation in Bitcoin Cash (a fork of the original Bitcoin). Next, we provide a formal proof of its security while respecting all the fundamental properties that an HDW should guarantee. Arcula brings for the first time identity-based addresses to the blockchain: By combining well-tested hierarchical key assignment schemes and certificates based on digital signatures, Arcula enables blockchain users to transmit coins by simply addressing receivers through their usernames or emails. All this while being formally secure against privilege escalation threats and with minimal overhead in the transaction costs. Furthermore, our design of Arcula also enables new capabilities for HDW: For instance, the possibility of dynamically modifying its hierarchy *e.g.*, by adding or removing users, and the possibility of constraining the distribution of the digital keys according to some temporal capabilities (*e.g.*, by distributing keys that will expire at the end of a specified period). Together, Arcula and all these properties unlock new use cases on the blockchain, in decentralized finance, and even for distributed or web-based financial institutions: Making it possible to register distributed loans, distributed promises or futures, and also to handle incoming payments on an entirely untrusted web-server securely.

¹ Arcula is the Latin word to define the small casket where ancient Romans used to store their jewels.

1.1 Our Contributions

We summarize our contributions below.

Syntax and Security model of HDW. We put forward the security definition of HDWs. Intuitively, an HDW should satisfy three properties: First, it deterministically generates its cryptographic keys (associated with users' coins) by starting from an initial seed provided by the user. As long as the user presents the same seed, she will be able to recover her keys, even in case of wallet loss. Second, an HDW also organizes the keys under an access hierarchy that combines groups of users with signing keys. The privileges of users depend on their level in the hierarchy. Users with higher privileges (*i.e.*, higher in the hierarchy), must be able to derive the keys of users on lower levels and, in turn, to sign messages (*i.e.*, transactions) on their behalf. Users on lower levels, however, should not be able to escalate their privileges along the hierarchy, not even when colluding with others. Third, an HDW should permit to generate every public key deterministically (*i.e.*, the address on which coins are transferred) of the wallet by starting from a *master public key*, without requiring any secret information in the process. As we shall see, this requirement enables a set of creative use cases for HDW (*e.g.*, public auditing of blockchain assets, and generation of new keys in untrusted environments). In more detail, let sk_i be the secret signing key of a user v_i of a hierarchy and let pk_i be the corresponding public key. Let sk_j and pk_j be, respectively, the secret and public keys associated with a descendant v_j of v_i (*i.e.*, a user with lower privileges in the hierarchy). We informally summarize the properties as follows:

Property 1 (Deterministic secret derivation). For each descendant v_j of v_i , the secret keys sk_j is deterministically generated by using the secret information of v_i . If v_j has the highest privileges in the hierarchy, then her secret information is generated from a user-provided seed.

Property 2 (Security against privilege escalation). For any set of colluding descendants of v_i , it is computationally infeasible to recover the secret key sk_i of v_i .

Property 3 (Public-key/Address derivation). The public key (address) pk_j of each descendant v_j of v_i is deterministically generated only using public information; the generation process does not require the secret key sk_i or any other secret information.

Arcula from Hierarchical Key Assignment. At its core, Arcula derives the keys of the users by relying on a deterministic version of hierarchical key assignment schemes (HKA): A provably secure process that takes as input a hierarchy of users and assigns a secret key to every user so that users with higher privileges can derive the keys of those below in the hierarchy. To the best of our knowledge, hierarchical key assignment schemes have never been used before to implement an HDW. One of our contributions is to connect these seemingly unrelated fields

of research. Hierarchical key assignment schemes provide several advantages: They are highly efficient and have been extensively studied in the past; they enable Arcula to implement arbitrarily complex hierarchies, integrate temporal capabilities into the wallet, and support the dynamic addition or removal of users to the hierarchy.

Implementation and Evaluation. We implemented Arcula, and we demonstrate that it can be used in the real world by showing how to send and receive funds on the Bitcoin Cash blockchain. Thanks to its design, Arcula is compatible with any blockchains whose scripting language permits the verification of a signature on an arbitrary message (*e.g.*, Ethereum). This is because users must prove ownership of identity-based addresses before spending coins by showing a certificate produced by the wallet’s administrator².

1.2 Applications

HDWs enable different use cases, inspired by both well established and innovative financial applications that specifically tackle the needs of enterprises, governments, and financial institutions.

Enterprises: In enterprises (*e.g.*, financial institutions, or exchanges), the hierarchy of a deterministic wallet might reflect the underlying chain of command or the subdivision in regions, departments, and teams. It allows managers to distribute funds among different branches and ensure fiscal responsibility. In particular, each branch can manage its funds but cannot spend those of other units. The deterministic generation of keys simplifies the management of secrets and guarantees a reliable recovery of the wallet, even in case of catastrophic loss. Cryptocurrency exchanges that manage the keys of hundreds of thousands of users might find this feature particularly useful: Through the HDW, they generate pairs of keys that take into account the hierarchy of users and then rely on the master seed to handle their recovery. Finally, the property of public-key/address derivation enables enterprises to comply with financial laws and regulations without jeopardizing the security of their infrastructure: *E.g.* it allows a (possibly untrusted) auditor to inspect the funds that they hold by deriving all the public keys in the wallet without knowing any secret information.

e-Commerce: An HDW is distinctly beneficial to an e-commerce marketplace. Marketplaces, such as Amazon, typically advertise and sell products to buyers. They also allow third-party vendors to do the same. Cryptocurrencies could help manage the payment flow to these vendors. When selling an item to a buyer, the marketplace generates a fresh payment address for each crypto-coin that it supports. As soon as the buyer transfers the required coins to one of the addresses and the blockchain confirms the transaction, the item gets shipped. The generation of fresh payment addresses leverages the properties of public-key

² In general, the administrator of the wallet is the highest privileged user in the hierarchy.

derivation: Since it does not require any private information, it can take place in an untrusted environment (*e.g.*, a web server exposed to the internet) and allows the e-commerce owner to derive the corresponding secret keys only when spending the funds (*e.g.*, by deriving them offline starting from an intermediate key of the wallet). Even if an attacker compromises the webserver, he will not discover any secret keys, and thus funds received before the attack remain safe and intact. Besides, public-key derivation allows buyers or auditors to check the authenticity of the payment addresses since anyone can generate them from the public key of the marketplace.

Decentralized Finance (DeFi): Decentralized Finance has recently started replacing many of the existing traditional financial tools (*e.g.*, loans, and futures) with open-source alternatives based on the blockchain and its smart contracts. With DeFi, HDW can unleash their full potential: The execution of smart contracts, indeed, cannot rely on any secret information as both the source code and the processing inputs are stored in the clear on the public blockchain. By combining HDW and public derivation, a DeFi smart contract can autonomously derive the public key of the recipient of a transaction (*e.g.*, a user that will receive interests on a loan, or the employee of a company that will receive a percentage of its shares). The process could take place on the blockchain and does not require manual interactions. In turn, HDW and DeFi lay the foundations of a modern, democratic, and decentralized financial world.

1.3 Our Technique

To provide a high-level description of Arcula, we begin by informally describing the security levels in which HDWs operate, ordered by the increasing trust requirements.

Untrusted Environment: This level is entirely untrusted. It refers, for example, to the executing environment of a payee that relies on public-key/address derivation (Property 3) to generate fresh addresses to deliver payments, or to an auditor that, generates the public address of a user (*e.g.*, a department of a company), to inspect the coins held by it.

Hot Environment: This level is semi-trusted. At this stage, the users can access their secret keys and derive those of their descendants. An attacker that compromises a user of the hot environment will compromise, in turn, only her descendants in the hierarchy (Property 2).

Cold Storage: This is the most trusted level of the security model, holding the seed used to generate every key within the wallet deterministically (Property 1). It typically corresponds to an offline location (*e.g.*, a hardware token used to instantiate the wallet) that is physically secured (*e.g.*, in a safe). An attacker that compromises the cold storage has full access to the wallet and every asset it holds.

Figure 1 provides an overview of the design of Arcula, our hierarchical deterministic wallet. Arcula starts from the seed to deterministically generate a master

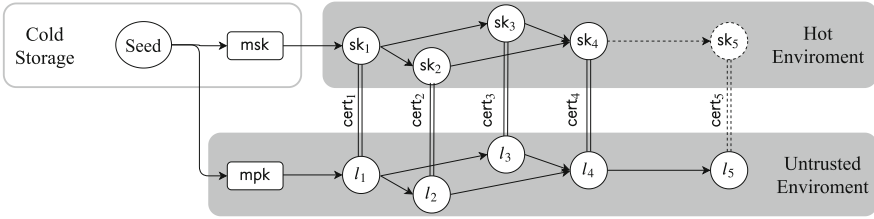


Fig. 1. A glance at the deterministic generation of secrets and identity-based public derivation within Arcula. The users l_1 to l_5 at the bottom of the figure create a hierarchy, encoded as a directed acyclic graph.

pair of secret and public keys (msk , mpk) (Property 1). After the initial instantiation, the seed and the master secret key can be safely moved to cold storage. The master public key uniquely identifies the wallet and will be used in the public key derivation process. In more detail, Arcula generates the public-key/address of i -th user of the hierarchy by concatenating the master public key mpk and her identity l_i (e.g., a numerical index or a bit string). As a result, Arcula achieves the public-key/address derivation (Property 3), and it can be executed, by design, in an untrusted environment. The master secret key msk allows, instead, to deterministically generate the secret keys sk_i corresponding to the users of the hierarchy. As previously mentioned, Arcula (hierarchically) generates the secret keys by leveraging a deterministic hierarchical key assignment. Such primitive assigns a derivation key to every user of the hierarchy, that, in turn, they will use to derive their own secret key sk_i and those of their descendants (Property 2). Note that in complex hierarchies (e.g., directed acyclic graphs), a user can have two (or multiple) parents (see l_4 and its two parents l_3 and l_2 of Fig. 1). The deterministic hierarchical key assignment allows each parent to compute the secret key of its child without knowing any of the secrets of the other parent. The private key generation should be executed in the context of the hot environment, as compromising a single user will lead to compromising every descendant.

While we have the hierarchical derivation on both the private and public side, the identity-based address and the secret key are completely unrelated. Because of that, Arcula explicitly associates secret keys to their corresponding identity-based addresses through a certificate cert_i signed by the master secret key msk , that links the identity l_i (top of Fig. 1) to the secret key sk_i generated by the deterministic key assignment scheme (bottom of Fig. 1). This is done by setting $\text{cert}_i = \text{Sign}_{\text{msk}}(\text{pk}_i || l_i)$ where pk_i is the real valid public-key of the secret key sk_i . Such a certificate must be revealed whenever a user wants to spend her coins to prove that it is the owner of a particular identity-based address (note that it can be verified just knowing mpk , the master public key of the wallet). Due to these certificates, Arcula can be used only with blockchains whose language permits the verification of signatures (in this case cert_i) on arbitrary messages (the pair $\text{pk}_i || l_i$).

Table 1. Comparison between Arcula and the existing state-of-the-art wallets.

	Security to Privilege Escalation (Property 2)	Public Key Derivation (Property 3)	Deterministic Generation (Property 1)	Hierarchy
BIP32 [17]	No	Yes	Yes	Tree
Hardened BIP32 [17]	Yes	No	Yes	Tree
Gutoski and Stebila [13]	No	Bounded	Yes	Tree
Fan <i>et al.</i> [10]	No	Yes	Yes	Tree
Poulami <i>et al.</i> [6]	–	–	–	No
Goldfeder <i>et al.</i> [12]	–	–	–	No
Gennaro <i>et al.</i> [11]	–	–	–	No
Dikshit and Singh [9]	–	–	–	No
Arcula (Section 5)	Yes	Yes	Bounded	DAG

The identity-based approach that we realized with Arcula provides several advantages: First, it solves the problem of distributing a public key for each user of the hierarchy. Also, it allows for generating an unbounded number of addresses for receiving transactions. On the other hand, Arcula relies on certificates signed by the master secret key to associate the secret key of a user with their identity, to which the transaction was addressed. Users only require these certificates when they sign a transaction for the first time: This means that their creation can be delayed until that moment and that it can occur entirely offline (dashed links sk_5 and $cert_5$ of user 5 in Fig. 1).

2 Related Work

Bitcoin Improvement Proposal 32 (BIP32) defines the state of the art implementation of hierarchical deterministic wallets [17]. In short, let g be the generator point of an Elliptic Curve. A private key sk_i is associated with its public key $pk_i = g^{sk_i}$. Let H be a hash function; the descendants' private keys sk_j and public keys pk_j are defined as:

$$sk_j = H(pk_i || j) + sk_i, \quad pk_j = g^{sk_j} = g^{H(pk_i || j) + sk_i} = g^{H(pk_i || j)} \cdot pk_i \quad (1)$$

Equation (1) satisfies the properties of deterministic generation and public derivation (Properties 1 and 3). However, there is a privilege escalation vulnerability because, by using the descendant private key sk_j and the parent public key pk_i , it is possible to recover the parent private key sk_i by computing $sk_i = sk_j - H(pk_i || j) \bmod q$. In other words, compromising any node leads to compromising the entire wallet. Such attack has been discussed extensively in [4, 5, 13, 17].

BIP32 addresses this issue by designing a *hardened* key derivation method that generates a descendant private key sk_j^h as follows: $\text{sk}_j^h = \text{H}(\text{sk}_i \| i) + \text{sk}_i \bmod q$. The hardened derivation solves the privilege escalation vulnerability but loses the public key derivation (*i.e.*, trades Property 3 for Property 2). Generating a hardened public key pk_j^h now requires the parent secret key sk_i (*i.e.*, $\text{pk}_j^h = g^{\text{sk}_j^h} = g^{\text{H}(\text{sk}_i \| j) + \text{sk}_i} = g^{\text{H}(\text{sk}_i \| j)} \cdot \text{pk}_i$).

In Table 1 we compare Arcula, our hierarchical deterministic wallet, with (hardened) BIP32 and the wallets present in the literature. BIP32 does not satisfy the security to privilege escalation property. The hardened version of BIP32, instead, fails in deriving public keys without requiring additional secrets. Gutoski and Stebila [13] propose an HDW strengthens the security of BIP32. Their design splits each secret key into n shares, distributed to the descendants of the user; reconstructing the secret key requires at least m shares. This solution provides weaker security than Property 2, because m colluding descendants of a user can recover the original secret key (as opposed to preventing any set of colluding descendants from escalating their privileges). In addition, they support Property 3 by publishing the public keys of all the users in the wallet. They do not allow the generation of fresh public keys, and their derivation is bounded to the number of published keys. Fan *et al.* [10] develop an HDW based on Schnorr signatures and trapdoor hash functions that enables the users to sign new transactions without accessing their private keys. A generic user can sign transactions on behalf of its descendants only after authorization by the root of the hierarchy that needs to reveal her the master private trapdoor key. As a result, any authorized user can sign new transactions on behalf of its descendants and all the users of the hierarchy. Compromising a single authorized user leads to revealing every secret stored in the wallet—the cold storage and hot environment of our threat model overlap, and the scheme is not secure against privilege escalation. Poulami *et al.* [6] provide a formal definition of non-hierarchical deterministic wallets and show a set of modifications that make ECDSA-based deterministic wallets provably secure. Goldfeder *et al.* [12] and subsequently Gennaro *et al.* [11] propose a non-hierarchical deterministic wallet where the secret key is shared among n parties, and at least t of them are required to sign a transaction. Dikshit and Singh [9] extend the threshold-based ECDSA signatures to assign different weights to the participants of the protocol. These works deal with non-hierarchical deterministic wallets, and they do not aim at achieving Properties 2 and 3 (depending on the hierarchical structure of the wallet).

Arcula, on the other hand, is the only solution secure against privilege escalation that, at the same time, supports a complex access hierarchy (*e.g.*, a directed acyclic graph (DAG)) and enables unbound public-key/address derivation. However, spending the coins addressed to one of the users requires creating a certificate that associates their identities to their keys. For this reason, Property 1 and, more precisely, spending coins within Arcula, is bound to the generation of a certificate, signed by the master secret key, that authorizes the users.

3 Preliminaries

Notation. Uppercase boldface letters (such as \mathbf{X}) are used to denote random variables, lowercase letters (such as x) to denote concrete values, calligraphic letters (such as \mathcal{X}) to denote sets, and sans serif letters (such as \mathbf{A}) to denote algorithms. Algorithms are modeled as (possibly interactive) Turing machines; if algorithm \mathbf{A} has access to some oracle \mathbf{O} , we often write $\mathcal{Q}_{\mathbf{O}}$ for the set of queries asked by \mathbf{A} to \mathbf{O} . For a string $x \in \{0, 1\}^*$, we let $|x|$ be its length; $|\mathcal{X}|$ represents the cardinality of the set \mathcal{X} . When x is chosen randomly in \mathcal{X} , we write $x \leftarrow_{\$} \mathcal{X}$. We write $y = \mathbf{A}(x)$ to denote a run of the algorithm \mathbf{A} on input x and output y ; if \mathbf{A} is randomized, y is a random variable and $\mathbf{A}(x; r)$ denotes a run of \mathbf{A} on input x and (uniform) randomness r . We write $y \leftarrow_{\$} \mathbf{A}(x)$ to denote a run of the randomized algorithm \mathbf{A} over the input x and uniform randomness. An algorithm \mathbf{A} is *probabilistic polynomial-time* (PPT) if \mathbf{A} is randomized and for any input $x, r \in \{0, 1\}^*$ the computation of $\mathbf{A}(x; r)$ terminates in a polynomial number of steps (in the input size). Throughout the paper, we denote by $\lambda \in \mathbb{N}$ the security parameter and we implicitly assume that every algorithm takes as input the security parameter. A function $\nu : \mathbb{N} \rightarrow [0, 1]$ is called *negligible* in the security parameter λ if it vanishes faster than the inverse of any polynomial in λ , i.e., $\nu(\lambda) \in \mathcal{O}(1/p(\lambda))$ for all positive polynomials $p(\lambda)$. We write $\text{negl}(\lambda)$ to denote an unspecified negligible function in the security parameter.

3.1 Signature Scheme

A signature scheme with message space \mathcal{M} is made of the following polynomial-time algorithms.

$\text{KGen}(1^\lambda)$: The randomized key generation algorithm takes the security parameter and outputs a secret and a public key (sk, pk).

$\text{Sign}(\text{sk}, m)$: The randomized signing algorithm takes as input the secret key sk and a message $m \in \mathcal{M}$, and produces a signature σ .

$\text{Vrfy}(\text{pk}, m, \sigma)$: The deterministic verification algorithm takes as input the public key pk , a message m , and a signature σ , and it returns a decision bit.

A signature scheme is correct if honestly generated signatures always verify correctly.

Definition 1 (Correctness of signatures). *A signature scheme $\Pi = (\text{KGen}, \text{Sign}, \text{Vrfy})$ with message space \mathcal{M} is correct if $\forall \lambda \in \mathbb{N}$ and $\forall m \in \mathcal{M}$, we have $\Pr[\text{Vrfy}(\text{pk}, m, \text{Sign}(\text{sk}, m))] = 1$, where $(\text{sk}, \text{pk}) \leftarrow_{\$} \text{KGen}(1^\lambda)$.*

For security, we are interested in existential unforgeability, i.e., it must be infeasible to forge a valid signature on a new fresh message.

Definition 2 (Unforgeability of signatures). *A signature scheme $\Pi = (\text{KGen}, \text{Sign}, \text{Vrfy})$ is existentially unforgeable under chosen-message attacks if for all PPT adversaries \mathbf{A} :*

$$\Pr[\mathbf{G}_{\Pi, \mathbf{A}}^{\text{uf}}(\lambda) = 1] \leq \text{negl}(\lambda),$$

where $\mathbf{G}_{\Pi, A}^{\text{euf}}(\lambda)$ is the following experiment:

Setup: The challenger runs $(\text{sk}, \text{pk}) \leftarrow_s \text{KGen}(1^\lambda)$ and gives pk to A .

Query: The adversary has access to a signing oracle $\mathcal{O}_{\text{Sign}}(\cdot)$. On input m , the challenger computes and returns $\sigma \leftarrow_s \text{Sign}(\text{sk}, m)$. Let $\mathcal{Q}_{\text{Sign}}$ denote the messages queried to the signing oracle.

Forgery: The adversary outputs (m, σ) . If $m \notin \mathcal{Q}_{\text{Sign}}$, and $\text{Vrfy}(\text{pk}, m, \sigma) = 1$, output 1, else output 0.

3.2 (Deterministic) Hierarchical Key Assignment Scheme

A hierarchical key assignment scheme [2] assigns a set of cryptographic keys to a set of users in a hierarchy. The hierarchy, encoded as a directed acyclic graph $G = (V, E)$, represents the access rights of users. A path from a node v_i to a node v_j implies that the user v_i has higher privileges than v_j and can assume the same access rights of v_j . We define the set of descendants $\text{Desc}(v_i) = \{v_j \mid v_i \rightsquigarrow_w v_j\}$ of node v_j to be the set of nodes v_j such that there exists a direct path w from v_i to v_j in G . An efficient hierarchical key assignment scheme (HKA) enforces the access hierarchy while minimizing the number of keys distributed to the users.

Typically, HKA schemes assign the cryptographic secrets that they assign at random. Our goal, however, is to leverage an HKA at the core of our wallet, where each secret is deterministically derived from a seed provided by the user. When the hierarchy is static (as in the scope of this paper), fixing the randomness of an HKA algorithm ensures a deterministic generation of keys (the same randomness implies the same key to each node). Within Arcula, we fix the randomness of the HKA developed by Atallah *et al.* [2] to achieve this goal. Besides, in the full version of this work [8], we show how to extend Arcula to handle dynamic hierarchies. In a dynamic context, fixing the randomness does not help since there is no a priori knowledge about the relative position of one node to each other, and, in particular, one cannot make assumptions on the order through which they are added to (or removed from) the hierarchy. For this reason, we propose a modification of the HKA [2] that achieves deterministic generation even with dynamic hierarchies.

A Deterministic Hierarchical Key Assignment (DHKA) scheme with seed space \mathcal{S} is composed of the following polynomial-time algorithms:

Set $(1^\lambda, G, S)$: The deterministic setup algorithm takes as input the security parameter, a DAG $G = (V, E)$, and an initial seed $S \in \mathcal{S}$, and outputs two mappings: 1) a public mapping $\text{Pub} : V \cup E \rightarrow \{0, 1\}^*$, associating a public label l_i to each node v_i in G and a public information $y_{i,j}$ to each edge $(v_i, v_j) \in E$; 2) a secret mapping $\text{Sec} : V \rightarrow \{0, 1\}^\lambda \times \{0, 1\}^\lambda$, associating a secret information S_i and a cryptographic key x_i to each node v_i in G . (No secret information is associated to the edges).

Derive $(G, \text{Pub}, v_i, v_j, S_i)$: The deterministic derivation algorithm takes as input the access graph G , the public information Pub , a source node v_i , a target node v_j , and the secret information S_i of node v_i . It outputs the cryptographic key x_j associated to node v_j if $v_j \in \text{Desc}(v_i)$.

The correctness of a DHKA scheme requires that any user v_i should be able to derive, correctly, the secret key x_j of any user $v_j \in Desc(v_i)$ lower in the hierarchy.

Definition 3 (Correctness of DHKA). A DHKA $\Pi = (\text{Set}, \text{Derive})$ with seed space \mathcal{S} is correct if for every DAG $G = (V, E)$, $\forall \lambda \in \mathbb{N}$, $\forall v_i \in V$, $\forall v_j \in Desc(v_i)$, $\forall S \in \mathcal{S}$, we have $\Pr[x_j = \text{Derive}(G, \text{Pub}, v_i, v_j, S_i)] = 1$, where $(\text{Pub}, \text{Sec}) = \text{Set}(1^\lambda, G, S)$, $(S_i, x_i) = \text{Sec}(v_i)$, and $(S_j, x_j) = \text{Sec}(v_j)$.

Instead, the security of DHKA requires that even if an attacker corrupts an arbitrary number of descendants of a node, he cannot distinguish its secret key from a uniformly random string. We adapt the definition originally defined by Atallah *et al.* [2] to account for the determinism in our scheme. We define the set of ancestors $Anc(v_i) = \{v_j \mid v_j \rightsquigarrow_w v_i\}$ of a node v_i to be the set of nodes v_j such that there exists a path w from v_j to v_i in G .

Definition 4 (Key Indistinguishability of DHKA). A DHKA $\Pi = (\text{Set}, \text{Derive})$ with seed space \mathcal{S} is key indistinguishable if for every PPT adversary \mathbf{A} and every DAG $G = (V, E)$:

$$\left| \Pr \left[\mathbf{G}_{\Pi, \mathbf{A}}^{\text{sk-ind}}(\lambda, G) = 1 \right] - \frac{1}{2} \right| \leq \text{negl}(\lambda),$$

where $\mathbf{G}_{\Pi, \mathbf{A}}^{\text{sk-ind}}(\lambda, G)$ is defined in the following way:

- Setup:** The challenger receives a challenge node $v^* \in V$ from the adversary \mathbf{A} . The challenger samples $S \leftarrow_s \mathcal{S}$, then runs $\text{Set}(1^\lambda, G, S)$, and gives the resulting public information Pub and G to \mathbf{A} . The challenger samples a random bit $b^* \leftarrow_s \{0, 1\}$: If $b^* = 0$, it returns to \mathbf{A} the cryptographic key x_{v^*} associated to node v^* ; otherwise, it returns a random key \tilde{x}_{v^*} of the corresponding length.
- Query:** The adversary has access to a corrupt oracle $\text{O}_{\text{Corr}}(\cdot)$. On input $v_i \notin Anc(v^*)$, the challenger retrieves $(S_i, x_i) = \text{Sec}(v_i)$ and sends S_i to \mathbf{A} .
- Guess:** The adversary outputs a bit $b \in \{0, 1\}$. If $b = b^*$ return 1; otherwise return 0.

4 Hierarchical Deterministic Wallet

A hierarchical deterministic wallet is composed of 5 algorithms ($\text{Set}, \text{DPub}, \text{DPriv}, \text{Sign}, \text{Vrfy}$): 1) Set deterministically instantiates the wallet by generating the public parameters pp and a set of the derivation key \mathbf{d}_i , one for user node v_i of the hierarchy. 2) DPriv and DPub are responsible of the derivation of signing and public keys (Properties 1 and 3). DPriv derives the signing key sk_j of a node v_j , descendent of v_i , by using the derivation key \mathbf{d}_i associated to v_i ; DPub derives the corresponding public key pk_j by using the only the public parameters pp . 3) Sign and Vrfy take inspiration from the standard signing and verification algorithms of a digital signature scheme. A hierarchical deterministic wallet $\Pi = (\text{Set}, \text{DPub}, \text{DPriv}, \text{Sign}, \text{Vrfy})$, defined over a seed space \mathcal{S} and message space \mathcal{M} is defined in the following way:

- Set**($1^\lambda, G, S$): The deterministic setup algorithm takes as input a security parameter, an access graph $G = (V, E)$, and an initial seed $S \in \mathcal{S}$, and outputs the public parameters \mathbf{pp} and a set of derivation keys $\{\mathbf{d}_i\}_{v_i \in V}$.
- DPub**(\mathbf{pp}, v_i): The deterministic public derivation algorithm takes as input the public parameters \mathbf{pp} , a target node v_i , and outputs the public key \mathbf{pk}_i associated to node v_i .
- DPriv**($\mathbf{pp}, \mathbf{d}_i, v_i, v_j$): The deterministic private derivation algorithm takes as input the public parameters \mathbf{pp} , the derivation key \mathbf{d}_i of node v_i , and a target node $v_j \in \text{Desc}(v_i)$, and outputs the secret key \mathbf{sk}_j associated to node v_j .
- Sign**(\mathbf{sk}_i, m): The randomized signing algorithm takes as input a message $m \in \mathcal{M}$, and a secret key \mathbf{sk}_i , and outputs a signature σ .
- Vrfy**(\mathbf{pk}_i, m, σ): The deterministic verification algorithm takes as input a public key \mathbf{pk}_i , a message m , and a signature σ , and outputs a decisional bit b .

A hierarchical deterministic wallet is correct if any user can derive the private and public key of its descendants and create a valid signature on behalf of them. This means that any node v_i can derive the signing key \mathbf{sk}_j of any node $v_j \in \text{Desc}(v_i)$ and produce, in turn, a valid signature σ on behalf of v_j (*i.e.*, that passes the verification process against the public key \mathbf{pk}_j obtained through public key derivation).

Definition 5 (Correctness of HDW). *A hierarchical deterministic wallet $\Pi = (\text{Set}, \text{DPub}, \text{DPriv}, \text{Sign}, \text{Vrfy})$, with seed space \mathcal{S} and message space \mathcal{M} , is correct if for every DAG $G = (V, E)$, $\forall v_i, v_j \in V$, $\forall v_j \in \text{Desc}(v_i)$, $\forall S \in \mathcal{S}$, $\forall m \in \mathcal{M}$ the following condition holds:*

$$\Pr [\text{Vrfy}(\mathbf{pk}_j, m, \text{Sign}(\mathbf{sk}_j, m)) = 1] \geq 1 - \text{negl}(\lambda),$$

where $(\mathbf{pp}, \{\mathbf{d}_i\}_{v_i \in V}) = \text{Set}(1^\lambda, G, S)$, $\mathbf{sk}_j = \text{DPriv}(\mathbf{pp}, \mathbf{d}_i, v_i, v_j)$, and $\mathbf{pk}_j = \text{DPub}(\mathbf{pp}, v_j)$.

The security of a hierarchical deterministic wallet draws inspiration from existentially unforgeable signatures. We allow an attacker to corrupt an arbitrary number of users in the hierarchy—by corrupting a user; the attacker implicitly corrupts also all her descendants. In addition, the attacker also has access to a signing oracle that returns signatures on arbitrary messages from any uncorrupted node. We challenge the attacker to forge a signature for a new message on behalf of an uncorrupted node.

Definition 6 (Hierarchical existential unforgeability of HDW). *A hierarchical deterministic wallet is hierarchically existentially unforgeable under chosen-message attacks if for every DAG $G = (V, E)$ and PPT adversary \mathbf{A} the following condition holds:*

$$\Pr [\mathbf{G}_{\Pi, \mathbf{A}}^{\text{heuf}}(\lambda, G) = 1] \leq \text{negl}(\lambda),$$

where experiment $\mathbf{G}_{\Pi, \mathbf{A}}^{\text{heuf}}(\lambda, G)$ is defined in the following way:

Setup: The challenger samples a random $S \leftarrow_{\$} \mathcal{S}$ and executes $(\mathbf{pp}, \{\mathbf{d}_i\}_{v_i \in V}) = \text{Set}(1^\lambda, G, S)$. It gives the public parameters \mathbf{pp} to \mathbf{A} .

Query: *The adversary A has access to the following oracles:*

$\mathcal{O}_{\text{Corr}}(\cdot)$: *On input $v_i \in V$, the challenger answers by giving \mathbf{d}_i to A. Let $\mathcal{Q}_{\text{Corr}}$ denote the set of nodes v_i that A corrupted, including their descendants $\text{Desc}(v_i)$.*

$\mathcal{O}_{\text{Sign}}(\cdot, \cdot)$: *On input $(m, v_i) \in \mathcal{M} \times V$, the challenger returns $\sigma \leftarrow \text{Sign}_{\text{sk}_i}(m)$ where $\text{sk}_i = \text{DPriv}(\text{pp}, \mathbf{d}_0, v_0, v_i)$. Let $\mathcal{Q}_{\text{Sign}}$ denote the pairs (m, v_i) for which A queried the oracle $\mathcal{O}_{\text{Sign}}$.*

Forgery: *A outputs a forgery (v_i, m, σ) . If $\text{Vrfy}_{\text{pk}_i}(m, \sigma) = 1$ where $\text{pk}_i = \text{DPub}(\text{pp}, v_i)$ and $v_i \notin \mathcal{Q}_{\text{Corr}}$, $(m, v_i) \notin \mathcal{Q}_{\text{Sign}}$, return 1; otherwise return 0.*

5 Constructing Arcula from DHKA and Signatures

This section details our construction based on deterministic hierarchical key assignment schemes (DHKA) and digital signatures.

Construction 1 *Let $\Gamma = (\text{Set}_\Gamma, \text{Derive}_\Gamma)$ and $\Sigma = (\text{KGen}_\Sigma, \text{Sign}_\Sigma, \text{Vrfy}_\Sigma)$ be respectively a DHKA and a signatures signature scheme. We build Arcula in the following way:*

$\text{Set}(1^\lambda, G, S)$: *On input the security parameter, a DAG $G = (V, E)$, and a seed $S \in \mathcal{S}$ the algorithm proceeds as follows:*

1. *Compute $(\text{Pub}, \text{Sec}) = \text{Set}_\Gamma(1^\lambda, G, S)$.*

2. *For each node $v_i \in V$:*

(a) *Let $(S_i, x_i) = \text{Sec}(v_i)$ and set $\mathbf{d}_i = S_i$.*

(b) *$(\overline{\text{sk}}_i, \overline{\text{pk}}_i) = \text{KGen}_\Sigma(1^\lambda; x_i)$.*

3. *Output $\text{pp} = (G, \text{Pub}, \{\text{cert}_i\}_{v_i \in V}, \overline{\text{pk}}_0)$ and $\{\mathbf{d}_i\}_{v_i \in V}$ where $\text{cert}_i \leftarrow \text{Sign}_\Sigma(\overline{\text{sk}}_0, (\overline{\text{pk}}_i, l_i))$ for $v_i \in V$, and $l_i = \text{Pub}(v_i)$.³*

$\text{DPub}(\text{pp}, v_j)$: *On input the public parameters $\text{pp} = (G, \text{Pub}, \{\text{cert}_i\}_{v_i \in V}, \overline{\text{pk}}_0)$ and a node $v_j \in V$, the algorithm returns $\text{pk}_j = (\overline{\text{pk}}_0, l_j)$ where $l_j = \text{Pub}(v_j)$.*

$\text{DPriv}(\text{pp}, \mathbf{d}_i, v_i, v_j)$: *On input the public parameters $\text{pp} = (G, \text{Pub}, \{\text{cert}_i\}_{v_i \in V}, \overline{\text{pk}}_0)$, the derivation key $\mathbf{d}_i = S_i$, and two nodes $v_i, v_j \in V$ such that $v_j \in \text{Desc}(v_i)$, the algorithm runs $x_j = \text{Derive}_\Gamma(G, \text{Pub}, v_i, v_j, S_i)$ and $(\overline{\text{sk}}_j, \overline{\text{pk}}_j) = \text{KGen}_\Sigma(1^\lambda; x_j)$. Finally, it returns $\text{sk}_j = (\overline{\text{sk}}_j, \overline{\text{pk}}_j, \text{cert}_j)$.*

$\text{Sign}(\text{sk}_i, m)$: *On input a signing key $\text{sk}_i = (\overline{\text{sk}}_i, \overline{\text{pk}}_i, \text{cert}_i)$ and a message m , the algorithms returns $\sigma = (\overline{\text{pk}}_i, \sigma', \text{cert}_i)$ where $\sigma' \leftarrow \text{Sign}_\Sigma(\overline{\text{sk}}_i, m)$.*

$\text{Vrfy}(\text{pk}_i, m, \sigma)$: *On input a public key $\text{pk}_i = (\overline{\text{pk}}_0, l_i)$, a message m , and a signature $\sigma = (\overline{\text{pk}}_i, \sigma', \text{cert}_i)$, the algorithms returns 1 if $\text{Vrfy}_\Sigma(\overline{\text{pk}}_0, (\overline{\text{pk}}_i, l_i), \text{cert}_i) = 1$ and $\text{Vrfy}_\Sigma(\text{pk}_i, m, \sigma') = 1$; otherwise it returns 0.*

The correctness of the scheme comes directly from the correctness of the underlying primitives. As for security, we establish the following result, whose proof appears in the full version [8]

³ The value l_i is the public label (binary string) associated by the DHKA to the node v_i . Without loss of generality we can assume that $l_i = v_i$ (the public label l_i is just the node number v_i).

Theorem 1. *Let $\Gamma = (\text{Set}_\Gamma, \text{Derive}_\Gamma)$ and $\Sigma = (\text{KGen}_\Sigma, \text{Sign}_\Sigma, \text{Vrfy}_\Sigma)$ be respectively a deterministic hierarchical key assignment and a signature scheme. If Γ is key indistinguishable (Definition 4) and Σ is existentially unforgeable (Definition 2), then the HDW II from Construction 1 is hierarchically existentially unforgeable (Definition 6).*

Security level of Arcula. Arcula’s public and secrets values must be correctly administered with respect to the HDW architecture model that we defined in Sect. 1.3 (see Fig. 1 for a graphical overview). As mentioned, the *Cold Storage* is where, besides the seed, the master secret key msk must be stored. In Arcula such master secret key correspond to the derivation key d_0 (and the related signing key sk_0) of the root (*i.e.*, $\text{msk} = \text{d}_0$). Indeed, an attacker that compromises d_0 can derive the signing key sk_i of every node, or generates fake certificates that would allow him to spend coins of any identity-based address of the wallet. On the other hand, the *Untrusted Environment* contains any public information that does not put in danger the security of the wallet. Naturally, as depicted in Fig. 1, the master public key mpk lies in this environment since it allows any third party to derive the public-keys/addresses of the hierarchy. Arcula’s master public key is simply the public signing key $\overline{\text{pk}}_0$ of the root (*i.e.*, $\text{mpk} = \overline{\text{pk}}_0$). Indeed, the public derivation requires just the concatenation of $\overline{\text{pk}}_0$ and the identifier l_i of node v_i . All the values $\{(\text{sk}_i, \text{d}_i)\}$ of any node v_i lie in the *Hot Environment*. Compromising the secrets of node v_i leads to compromising all the secrets of its descendants, but none of the other nodes. Lastly, we mention that the certificates $\{\text{cert}_i\}$ they can be published (or even stored on an external server). Even if an adversary knows all the certificates $\{\text{cert}_i\}$ of the wallet, it can not spend the coins of a target user without knowing its secret key $\overline{\text{sk}}_i$.

Extensions. By building Arcula on top of deterministic HKA schemes, our design also inherits some of their more interesting properties. As an example, in the full version of the paper [8], we show: 1) How it is possible to extend Arcula to handle dynamic hierarchies by leveraging the dynamic version of the HKA developed by Atallah *et al.* [2, Section 6] (note that some of the procedures to modify the hierarchy of the HKA proposed in [2, Section 6] are randomized. In order to work with Arcula, we make these procedures deterministic by leveraging a PRF. Intuitively we generate the randomness required by executing the PRF) and, 2) how to incorporate temporal capabilities (*e.g.*, possibility to use cryptographic keys during a given period only) into Arcula by relying on the work of De Santis *et al.* [7].

6 Arcula in the Real World

In this section, we present how Arcula performs in the real world, and we show how to use it to send and receive funds on the Bitcoin Cash blockchain. We stress that Arcula can be used in any blockchain system that enables the verification of signatures of an arbitrary message (*e.g.*, Ethereum) since it requires the verification of a certificate cert_i .

6.1 Technical Implementation

Our open-source implementation of Arcula is available online.⁴ We instantiate the underlying DHKA leveraged by Arcula with the pseudorandom function $F_k(x) = H(k||x)$ (where $H(x)$ is the hash function $\text{SHA3-256}(x)$) and the authenticated AES256 with Galois/Counter Mode (GCM) as the symmetric encryption scheme. We generate a hierarchical deterministic wallet based on the tree defined in BIP43 and BIP44 [14, 15], where the keys to different crypto-coins correspond to different subtrees, and each branch of the subtrees is a chain associated with a single account that contains multiple receiving addresses. We obtain an initial seed S of 512 bits by following the specification of BIP39 [16] that generates a seed from a random mnemonic sequence. We generate the wallet that we use in our tests by fixing the randomness of the mnemonic generation process to the result of the operation `H(correct horse battery staple)`.

6.2 Arcula in Bitcoin Cash

Transactions in Bitcoin Cash. A Bitcoin (Cash) transaction is a cryptographically signed statement that transfers some coins from a sender to a receiver. The sender of the coins signs the transaction through her secret key to spending, in turn, the coins destined to the corresponding public key. Every transaction specifies a locking and an unlocking script. These scripts respectively state the necessary conditions to spend, in a future transaction, the coins being transferred (*i.e.*, their locking condition) and provide the information required to redeem them (*i.e.*, to unlock them as a result of a past transaction). Both scripts are written through a stack-based language that allows simple mathematical operations, stack manipulations and enables simple cryptographic primitives (*i.e.* computing the result of a hash function and verifying a signature).

A typical Bitcoin locking script specifies the address of the receiver (usually through the hash of its public key) and requires him to provide a valid signature to redeem the coins being transferred. More in detail, the locking and unlocking scripts of a standard Bitcoin transaction are defined as follows. Uppercase monospace words indicate operations of the Bitcoin scripting language, while angular brackets enclose variable inputs.

Locking: `OP_DUP OP_HASH160 <H(pk)> OP_EQUALVERIFY OP_CHECKSIG`

Unlocking: `< σ > <pk>`

Together, these scripts ensure that the public key pk provided in the unlocking script is the pre-image of the hash $H(pk)$ (the Bitcoin address) contained in the locking script; then, verify the validity of the transaction signature σ under the public key pk .

Every transaction devolves a small amount of fees to the system to incentivize its inclusion in the next block of the chain. Fees are usually measured in coins per byte, and, for this reason, the size of a transaction on the Bitcoin wire protocol

⁴ Available at <https://github.com/aldur/Arcula>.

is directly related to the amount of fees that it should pay to be included in the blockchain. In particular, the length of the locking and unlocking scripts influences directly the final transaction cost. Because of this, BIP16 [1] proposes the pay to script hash mechanism (P2SH), that aims at minimizing the size of the locking script, such that the sender of the transaction will pay its associated fees⁵. The intuition is that instead of specifying the full locking script, the users can constrain the coins of a transaction by locking them to the hash of the original script; then, in the unlocking script, they can provide both the pre-image of the hash, *i.e.*, the full locking script, and its required inputs. Hence, the P2SH locking script is constant (*i.e.*, hash verification) while the unlocking one is the concatenation of the standard locking and unlocking scripts while the locking is constant.

Arcula's Transactions. In Arcula, we identify the nodes of our wallet v_i according to the master public key $\text{mpk} = \overline{\text{pk}}_0$ and to their public label l_i . For this reason, an Arcula address is simply the concatenation of the byte representations of these values that we encode in the locking script. The unlocking script, on the other hand, contains the certificate $\text{cert}_i \leftarrow \text{Sign}_{\Sigma}(\overline{\text{sk}}_0, (\overline{\text{pk}}_i, l_i))$ and associating the signing public key $\overline{\text{pk}}_i$ to the node v_i with label l_i , and a signature σ of the transaction verifiable through the public signing key $\overline{\text{pk}}_i$. With Arcula, the locking and the unlocking scripts respectively become:

Locking: `OP_DUP OP_TOALTSTACK < l_i > OP_CAT < mpk > OP_CHECKDATASIGVERIFY
OP_FROMALTSTACK OP_CHECKSIG`
Unlocking: `< σ > < certi > < $\overline{\text{pk}}_i$ >`

The two scripts: 1) Verify that the certificate cert_i is a valid signature of the message $(\overline{\text{pk}}_i, l_i)$ under the master public key mpk ; 2) verify the validity of the transaction signature σ under the signing public key $\overline{\text{pk}}_i$. In particular, the locking script checks the validity of the certificate cert_i through the operation `OP_CHECKDATASIGVERIFY`, which allows the stack-based scripting language to validate a signature of an arbitrary message (the concatenation of mpk and l_i obtained through the operation `OP_CAT`). To test Arcula, we focus, as an example, on Bitcoin Cash (the original Bitcoin implementation does not support such operation). We first create a transaction that locks 0.5 BCH (the Bitcoin Cash crypto-coin) to a node of our wallet of Sect. 6.1, identified through the master public key mpk (also in the locking script) and the integer label 3. Next, we redeem the coins through a second transaction that provides the transaction signature σ computed using the signing key $\overline{\text{sk}}_i$, an appropriate certificate cert_i signed by the master secret key, and the public signing key $\overline{\text{pk}}_i$ ⁶. We create both the signature and the certificate through the ECDSA signatures scheme on the `secp256k1` elliptic curve used in Bitcoin, and we encode the integer label l_i of the node v_i with 4 bytes.

⁵ Another advantage of P2SH is that it hides the details of the locking script until the users redeem the coins sent by the transaction.

⁶ The transcripts of the transactions are available, respectively, at <https://bit.ly/2UI62tt> and <https://bit.ly/2UoQNGI>.

Table 2. The script bytes sizes of a transaction to a standard Bitcoin address and to an Arcula address. Between brackets the sizes when pay to script hash (P2SH) is in place [1].

Address type	Locking Script	Unlocking Script	Total
Standard (with P2SH)	24 (22)	106 (130)	130 (152)
Arcula (with P2SH)	43 (22)	179 (222)	222 (244)

Transaction Costs. Table 2 reports the sizes, in bytes, of the locking and unlocking scripts of standard Bitcoin transactions and to addresses of our wallet. Every operation of the stack-based scripting language is encoded with a single byte; a standard Bitcoin address is the result of a hash function that outputs 20 bytes; the ECDSA signature and the public key in the unlocking script require, respectively, 73 and 33 bytes. By summing these values up, we find that the locking script of a transaction (without P2SH) to a standard Bitcoin address is 24 bytes long (4 script operations plus the receiver address). In contrast, the unlocking scripts take 106 bytes (the ECDSA signature and its associated public key). In Arcula, on the other hand, the locking script encodes 6 operations, the identifier of a node (that we encode with 4 bytes), and the cold storage public key (33 bytes, as opposed to its 20 bytes hash), for a total of 43 bytes. The unlocking script, instead, contains two ECDSA signatures (one for the transaction and one for the certificate) and the signing public key; as a result, it is 179 bytes long. Overall, the size of the locking and unlocking scripts for a transaction to an Arcula address is 222 bytes, 70% longer than the standard address counterparts. On the other hand, when P2SH is used, the Arcula’s size overhead drops to 60%.

In many cases, the benefits that arise with Arcula justify the increase in the transaction cost. An e-commerce marketplace, as an example, can leverage Arcula’s public key derivation to dynamically derive new addresses (*e.g.*, one for each product of her catalog) in an entirely untrusted environment (*e.g.*, an online web-server) while keeping every signing keys at rest in trusted storage. As a result, the provider obtains the flexibility of handling incoming payments on dynamic addresses and minimizes the risk of losing the coins associated with them. When compared with the financial costs associated with this risk, the additional fees required by the Arcula transactions are negligible. The public key derivation also brings other significant benefits. Many financial regulations require, indeed, companies to be accountable for all the payments that they receive. With Arcula, an auditor can reach this goal by merely inspecting the blockchain while looking for any address that contains the master public key mpk that identifies the company. Finally, many enterprises leverage m -of- n signatures, where redeeming a transaction requires m valid signatures among n authorized public keys. Their goal is to enforce the company’s internal structure (*e.g.*, so that either managers or employees can sign transactions) or divide the responsibility of spending coins evenly. The unlocking scripts of m -of- n transactions have considerable size: They contain m signatures and n public keys.

By leveraging Arcula and enforcing an appropriate hierarchy that reflects their internal structure, these companies could reduce the size of the unlocking scripts to only two signatures (the transaction signature and the certificate) and two public keys (the master and signing public keys).

6.3 Optimizations and Compatibility with Bitcoin

The current implementation of Arcula does not require any modification to the underlying protocols and blockchains whose scripting languages allow the verification of signatures of arbitrary messages. Nevertheless, we also propose a set of optimizations that, through minimal modifications to these protocols, reduce both the cost of transactions to Arcula addresses and the amount of storage required on the blockchain. We begin by noting that any authorization certificate cert_i can be used more than once. For this reason, the first optimization that we propose is to *cache* the certificate cert_i as soon as it appears for the first time in an unlocking script. Then, any subsequent transaction signed by $\overline{\text{sk}}_i$ could specify a pointer to the certificate (*e.g.*, with a shorter hash) instead of the certificate itself and, in turn, reduce the size of the unlocking script. As an example, by pointing to the certificate with a 20 bytes hash, we would reduce the size of the Arcula locking and unlocking scripts to be roughly 20 bytes longer than their traditional counterparts. Implementing this optimization requires a new operation in the scripting language to retrieve and verify the certificate.

On the other hand, if we allow for more complex modifications, we can change the signature scheme of the underlying protocols to reduce these space requirements to their optimal value further—a single signature per transaction. Arcula can be implemented with a single signature by leveraging a sanitizable signature scheme [3], *i.e.* a scheme where an authorized party can modify a fraction of the message signed without interacting with the original signer. The intuition is to combine the certificates with the signatures that authorize transactions: Now, the certificate of user v_i that associates her to the signing public key $\overline{\text{pk}}_i$ also includes an additional modifiable portion that will be filled with the transaction details. To spend their coins, the users leverage their sanitizable key to replace the blank transaction with the details they intend to sign.⁷ In their work, Ateniese *et al.* [3] show how to construct a sanitizable signature scheme by combining any signature scheme with a chameleon hash function. This construction would allow Arcula to be used with the traditional Bitcoin blockchain by implementing the sanitizable signatures on top of the ECDSA signature scheme that it already uses. In addition, it would not change the expressiveness of the Bitcoin scripting language: Instead of enabling the verification of signatures on arbitrary messages, it would simply extend the signature verification protocol to account for the certificate embedded in the sanitized signatures.

⁷ The sanitizable keys can be hierarchically deployed by leveraging a second instance of DHKA.

6.4 Unlinkability of Transactions

Individual users of hierarchical deterministic wallets are typically not interested in public key derivation. Unlike enterprises and e-commerce marketplaces, for instance, they simply rely on HDW to recover their keys in case of hardware failure or catastrophic loss. On the other hand, they are often interested in achieving the unlinkability of their transactions, *i.e.* in making sure that multiple transactions sent to their wallet can not be correlated together by an observer that passively monitors the blockchain. However, an HDW that satisfies the public derivation property (Property 3), inevitably reveals the relation between the public keys of the wallet, making it impossible to achieve any privacy notion.

Arcula allows users to trade the public the derivation of public keys in an untrusted setting for the ability to receive payments on uncorrelated pseudonyms. In more detail, these users can ignore the identity-based public derivation that Arcula provides and identify the nodes of the wallet by only their public key $\overline{\text{pk}}_i$. On the blockchain, they can receive standard transactions (costing standard transaction fees) on the public key $\overline{\text{pk}}_i$ and then sign new transactions to redeem the coins through the corresponding private key $\overline{\text{sk}}_i$. Each key pair $(\overline{\text{sk}}_i, \overline{\text{pk}}_i)$ is generated through the underlying DHKA scheme that is secure under key indistinguishability. As a consequence, every two distinct pair $(\overline{\text{sk}}_i, \overline{\text{pk}}_i)$ and $(\overline{\text{sk}}_j, \overline{\text{pk}}_j)$ looks random and independent. As a result, Arcula provides a provably secure alternative to the hardened mode of BIP32: Individual users can generate as many pseudonyms as they need by branching or deepening the DAG that encodes their hierarchy and then leverage the DHKA to generate keys and reliably recover them in case of loss. Note that this modified version of Arcula does not require validation of signatures of arbitrary messages, enabling its usage with any blockchain system, including Bitcoin.

7 Conclusions

In this work, we presented Arcula, a new hierarchical deterministic wallet (HDW) that brings identity-based addresses to the blockchain and is secure against privilege escalation. We leveraged a deterministic hierarchical key assignment (DHKA) scheme to generate the set of cryptographic keys at the core of our wallet. As a result, an attacker that compromises an arbitrary number of users in the hierarchy can not escalate his privileges and compromise other users higher in the hierarchy. Our wallet allows us to dynamically derive new addresses for receiving payments in an entirely untrusted environment, recover every cryptographic key from an initial seed provided by the user, and spend coins on behalf of users lower in the hierarchy. Our design of Arcula considers the legacy and future requirements of modern blockchains. In particular, Arcula is independent of the underlying signature scheme, and it works on top of any protocol that allows the verification of signatures on an arbitrary message (*e.g.*, Bitcoin Cash or Ethereum). For these reasons, we hope that the outcomes of this work will be twofold: To provide the secure and efficient hierarchical deterministic wallet that

we need today and to propose a future-proof design that supports the financial applications and tools of enterprises and companies at scale.

References

1. Andresen, G.: BIP16: pay to script hash (2012). <https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki>. Accessed 9 Nov 2020
2. Atallah, M.J., Blanton, M., Fazio, N., Frikken, K.B.: Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **12**(3), 1–43 (2009)
3. Ateniese, Giuseppe., Chou, Daniel H., de Medeiros, Breno, Tsudik, Gene: Sanitizable signatures. In: di Vimercati, Sabrina de Capitani, Syverson, Paul, Gollmann, Dieter (eds.) *ESORICS 2005*. LNCS, vol. 3679, pp. 159–177. Springer, Heidelberg (2005). https://doi.org/10.1007/11555827_10
4. Buterin, V.: Deterministic wallets, their advantages and their understated flaws (2013). <https://bitcoinmagazine.com/articles/deterministic-wallets-advantages-flaw-1385450276/>. Accessed 9 Nov 2020
5. Courtois, N.T., Valsorda, F., Emirdag, P.: Private key recovery combination attacks: on extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor RNG events (2014)
6. Das, P., Faust, S., Loss, J.: A formal treatment of deterministic wallets. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 651–668 (2019)
7. De Santis, A., Ferrara, A.L., Masucci, B.: New constructions for provably-secure time-bound hierarchical key assignment schemes. *Theor. Comput. Sci.* **407**(1–3), 213–230 (2008)
8. Di Luzio, A., Francati, D., Ateniese, G.: Arcula: A secure hierarchical deterministic wallet for multi-asset blockchains. arXiv preprint [arXiv:1906.05919](https://arxiv.org/abs/1906.05919) (2019)
9. Dikshit, P., Singh, K.: Efficient weighted threshold ECDSA for securing bitcoin wallet. In: *2017 ISEA Asia Security and Privacy (ISEASP)*, pp. 1–9. IEEE (2017)
10. Fan, Chun-I., Tseng, Yi-Fan., Su, Hui-Po., Hsu, Ruei-Hau, Kikuchi, Hiroaki: Secure hierarchical bitcoin wallet scheme against privilege escalation attacks. *Int. J. Inf. Secur.* **19**(3), 1–11 (2019). <https://doi.org/10.1007/s10207-019-00476-5>
11. Gennaro, Rosario., Goldfeder, Steven, Narayanan, Arvind: Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In: Manulis, Mark, Sadeghi, Ahmad-Reza, Schneider, Steve (eds.) *ACNS 2016*. LNCS, vol. 9696, pp. 156–174. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39555-5_9
12. Goldfeder, S., et al.: Securing bitcoin wallets via a new DSA/ECDSA threshold signature scheme (2015)
13. Gutoski, Gus, Stebila, Douglas: Hierarchical deterministic bitcoin wallets that tolerate key leakage. In: Böhme, Rainer, Okamoto, Tatsuaki (eds.) *FC 2015*. LNCS, vol. 8975, pp. 497–504. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_31
14. Palatinus, M., Rusnak, P.: BIP43: purpose field for deterministic wallets (2014). <https://github.com/bitcoin/bips/blob/master/bip-0043.mediawiki>. Accessed 9 Nov 2020
15. Palatinus, M., Rusnak, P.: BIP44: multi-account hierarchy for deterministic wallets (2014). <https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki>. Accessed 9 Nov 2020

16. Palatinus, M., Rusnak, P., Voisine, A., Bowe, S.: BIP39: Mnemonic code for generating deterministic keys (2013). <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>. Accessed 9 Nov 2020
17. Wuille, P.: BIP32: Hierarchical deterministic wallets (2012). <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>. Accessed 9 Nov 2020



Detecting Covert Cryptomining Using HPC

Ankit Gangwal¹(✉), Samuele Giuliano Piazzetta², Gianluca Lain²,
and Mauro Conti³

¹ TU Delft, Delft, The Netherlands
a.gangwal@tudelft.nl

² ETH Zürich, Zürich, Switzerland
{spiazzetta,gilain}@student.ethz.ch

³ University of Padua, Padua, Italy
conti@math.unipd.it

Abstract. Cybercriminals have been exploiting cryptocurrencies to commit various unique financial frauds. Covert cryptomining - which is defined as an unauthorized harnessing of victims' computational resources to mine cryptocurrencies - is one of the prevalent ways nowadays used by cybercriminals to earn financial benefits. Such exploitation of resources causes financial losses to the victims.

In this paper, we present our efficient approach to detect covert cryptomining on users' machine. Our solution is a generic solution that, unlike currently available solutions to detect covert cryptomining, is not tailored to a specific cryptocurrency or a particular form of cryptomining. In particular, we focus on the core mining algorithms and utilize Hardware Performance Counters (HPC) to create clean signatures that grasp the execution pattern of these algorithms on a processor. We built a complete implementation of our solution employing advanced machine learning techniques. We evaluated our methodology on two different processors through an exhaustive set of experiments. In our experiments, we considered all the cryptocurrencies mined by the top-10 mining pools, which collectively represent the largest share of the cryptomining market. Our results show that our classifier can achieve a near-perfect classification with samples of length as low as five seconds. Due to its robust and practical design, our solution can even adapt to zero-day cryptocurrencies. Finally, we believe our solution is scalable and can be deployed to tackle the uprising problem of covert cryptomining.

Keywords: Cryptocurrency · Machine learning · Mining · Profiling

1 Introduction

Cryptomining, or simply mining, is a process of validating and adding new transaction in the blockchain digital ledger for various cryptocurrency. It is an essential process to keep most of the cryptocurrencies running. Typically, mining is a

resource-intensive process that continuously performs heavy computations. Upon successful mining, miners receive newly generated cryptocurrencies as their remuneration. Usually, newer cryptocurrencies tend to pay a higher reward. Some cryptocurrencies, such as Monero, make mining feasible on the web-browsers that enable even layman users to participate in mining.

After the success of Bitcoin [40], many alternative cryptocurrencies (altcoins) have been introduced to the market. At the time of writing, there are over 2000 active cryptocurrencies [2]. The massive number of cryptocurrencies raises an enormous demand for mining. This demand continues to remain huge because mining, as mentioned before, is an inevitable operation to keep these virtual currency systems running. Such an immense demand for mining has attracted cyber-criminals [7, 18] to earn financial gains, who have already been exploiting cryptocurrencies to perform several types of financial crimes, e.g., ransomware [29].

Motivation: A genuine miner has to make an investment in hardware and bear the significant cost of electricity to run the mining hardware as well as cooling facilities [14]. Nevertheless, mining is not beneficial on personal expenditure (mainly, on electricity) unless mining is performed with specialized hardware [16]. However, mining can be very profitable if it is performed with “stolen” resources, e.g., through covert cryptomining, or simply cryptojacking. Cryptojacking is defined as an unauthorized use of the computing resources on a computer, tablet, mobile phone, or connected home device to mine cryptocurrencies.

Cybercriminals have made several ingenious attempts to spread cryptojackers in the form of malware [20], malicious browser extensions [12], etc.. by exploiting vulnerability [17], compromising third-party plug-ins [19], maneuvering misconfigurations [11], taking advantage of web-based hosting service [13], and so on. To evade intrinsic detection techniques (e.g., processor’s usage), some cryptojackers suspend their execution when the victim is using the computer [31], use “pop-under” windows to keep mining for a comparatively longer duration [8], and utilize legitimate processes of the operating system to mine [28]. Moreover, merely monitoring CPU load, etc.. is an ineffective strategy because of both false positives and false negatives [37].

To further aggravate the situation, cryptocurrency mining service (e.g., Coinhive [1], Crypto-Loot [3]) easily integrate into websites to monetize the computational power of their visitors. In fact, cryptojacking attacks exceeded ransomware attacks in 2018 and affected five times more systems as compared to ransomware [25]. According to Symantec’s report [10], almost double cryptominers were detected on consumer machines as compared to enterprise machines between October 2017 and February 2018 while the same volume of cryptominers was detected on consumer and enterprise machines between March 2018 and July 2018. Kaspersky’s report [15] shows that the total number of internet users who encountered cryptominers rose from 1.9 million in 2016–2017 to 2.7 million in 2017–2018. IBM X-Force Threat Intelligence Index 2019 [23] estimates that cryptojacking attacks increased by more than 4-times (~450%) from Q1 2018 to Q4 2018. SonicWall researchers [24] reported that cryptojacking attackers made 52.7 million cryptojacking hits during the first half of 2019. Such exploitation of

the computational resources causes financial damage - primarily in the form of increased¹ electricity bills - to the victims, who often discover the misuse when the damage has already been done.

On another side, the current state of cryptomining has been consuming a vast amount of energy. As a representative example, Bitcoin Energy Consumption Index was created to provide insight into this amount with respect to Bitcoin, Bitcoin network consumes electricity close to the total demand by Iraq, and a single Bitcoin transaction requires nearly 2.7 times the electrical energy consumed by 100,000 transactions on the VISA network [9]. Moreover, a recent study [39] has suggested that “Bitcoin usage could alone produce enough CO_2 emissions to push warming above $2^\circ C$ within less than three decades.” The current situation would further worsen with illegal/unauthorized/covert cryptomining. Finally, the abundance of the active cryptocurrencies raises the demand for a generic solution to detect covert cryptomining that does not focus on a particular cryptocurrency.

Contribution: In this paper, we focus on detecting covert cryptomining on users’ machine. The major contributions of this paper are as follows:

1. We propose an efficient approach to detect covert cryptomining. In particular, our approach uses HPC to profile the core of the mining process, i.e., the mining algorithms, on a given processor to accurately identify cryptomining in real-time. We designed our solution to be a generic one, i.e., it is not tailored to a particular cryptocurrency or a specific form of cryptomining.
2. We exhaustively assess the quality of our proposed approach. To this end, we designed six different experiments: (1) *binary* classification; (2) *currency* classification; (3) *nested* classification; (4) *sample length*; (5) *feature relevance*; and (6) *unseen miner programs*. For a thorough evaluation, we considered eleven distinct cryptocurrencies in our experiments. Our results show that our classifier can accurately classify cryptomining activities.
3. In the spirit of reproducible research, we make our collected datasets and the code publicly available².

Organization: The remainder of this paper is organized as follows. Section 2 presents a summary of the related works. We explain our system’s architecture in Sect. 3 and discuss its evaluation in Sect. 4. Section 5 addresses the potential limitations of our solution. Finally, Sect. 6 concludes the paper.

2 Related Works

HPC are special-purpose registers in modern microprocessors that count and store hardware-related activities. These activities are commonly referred to as

¹ A machine consistently performs heavy computations while it does cryptomining, which, in turn, continuously draws electricity.

² spritz.math.unipd.it/projects/cryptojackers/.

hardware *events*³. HPC are often used to conduct low-level performance analysis and tuning. HPC-based monitoring has very low-performance overhead, which makes it suitable even for latency-sensitive systems. Several works have shown the effectiveness of using HPC to detect generic malware [32, 46, 48], kernel-level rootkits [47], side-channel attacks [27], unauthorized firmware modifications [45], etc.

A general-purpose process classification may distinguish a browser application from a media player or one browser application from another browser application. In the former case, the nature of the applications is different while both the applications in the latter case have the same nature and perform the same operation of rendering pages. Cryptominers have the same nature (of mining), but they essentially perform very different underlying operations due to different proof-of-works, and they also require different compute resources (e.g., BTC⁴ mining is processor-oriented while XMR mining is memory-oriented). Hence, a comparison of our work with the general-purpose process classification methods falls out of the scope of this paper.

On another side, there are limited number of works on detecting cryptomining. Bonneau et al. [26] discuss open research challenges of various cryptocurrencies and their mining. Huang et al. [36] present a systematic study of Bitcoin mining malware and have shown that modern botnets tend to do illegal cryptomining. Gangwal et al. [33] use magnetic side-channel to detect cryptomining. Other works [37, 38, 41, 42, 44] focus particularly on browser-based mining. However, only a limited number of cryptocurrencies can be mined in the web-browsers. MineGuard [43] focuses on detecting cryptomining operations in the cloud infrastructure.

Our work is different from the state-of-the-art on the following dimensions: (1) our proposed solution is a generic solution that is not tailored to a particular cryptocurrency or a specific form (e.g., browser-based) of cryptomining on computers; and (2) we tested our solution against all the cryptocurrencies mined by the top-10 mining pools, which collectively represent the largest portion of the cryptomining business.

3 System Architecture

We elucidate the key concept behind our approach in Sect. 3.1, our data collection phase in Sect. 3.2, selection of cryptocurrencies in Sect. 3.3, and our classifier’s design in Sect. 3.4.

3.1 Fundamental Intuition of Our Approach

The task of cryptomining requires a miner to run the core Proof-of-Work (PoW⁵) algorithm repetitively to solve the cryptographic puzzle. At a coarse-grained

³ An *event* is defined as a countable activity, action, or occurrence on a device.

⁴ To refer to different cryptocurrencies, we use their standard ticker symbol. See Table 3 for acronyms and their corresponding cryptocurrencies.

⁵ We use the term “PoW” to represent different consensus algorithms.

level, some PoW algorithms are processor-oriented (e.g., BTC) while some are memory-oriented (e.g., XMR) due to their underlying design. At a fine-grained level, each PoW algorithm has its own unique mathematical/logical computations (or, in other words, the sequence of operations). Thus, each algorithm upon execution affects some specific *events* more as compared to other *events* on the processor. Consequently, when an algorithm is executed several times repetitively, the “more” affected *events* outnumber the other - relatively under affected - *events*. It means that a discernible signature can be built using the relevant *events* for a PoW algorithm. As a representative example, Fig. 1 depicts the variation in *events* while mining different cryptocurrencies and performing some common user-tasks. LTC, for instance, shows a more erratic pattern in cache-misses as compared to the other *events* that are affected during LTC mining. On the other hand, a Skype video call has more disparity in context-switches.

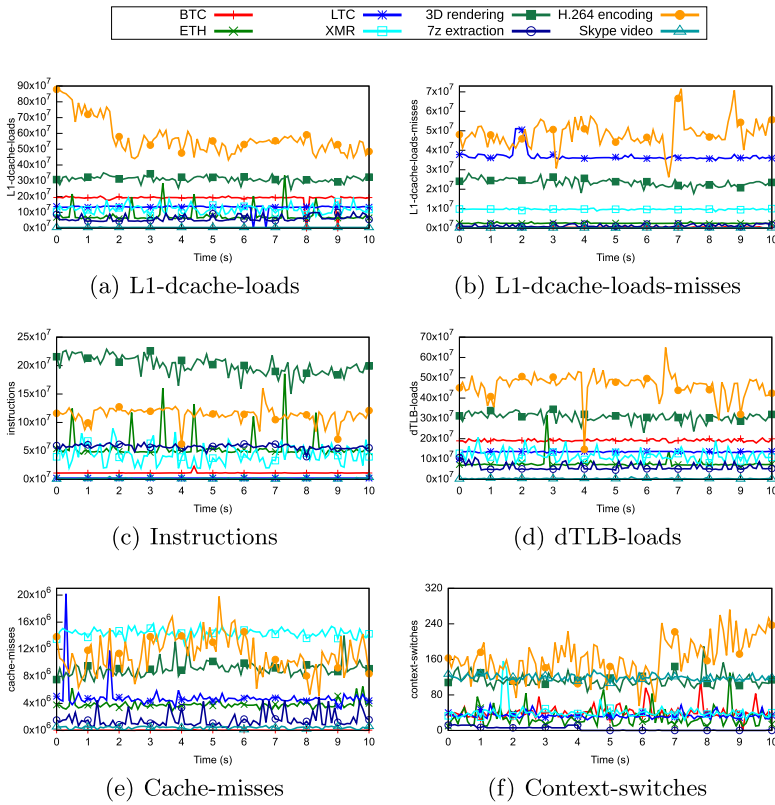


Fig. 1. A representative example of variation in *events* while mining different cryptocurrencies and performing some common user-tasks. HPC were polled every 100 ms. The line-points in the graphs do not represent data points and are merely used to make lines distinguishable.

In practice, there is a finite number of PoW algorithms upon which cryptocurrencies are established. So, we concentrate on the mining algorithms instead of individual currency in our solution. To this end, we use supervised machine learning (cf. Sect. 3.4) to construct signatures and build our classifier.

On another side, an adversary may attempt to circumvent such signature-based detection in the following ways: (1) by controlling/limiting the mining; or (2) by neutralizing the signatures. Limiting the mining would reduce the hashing rate, which would indeed make the mining less profitable. Whereas, to neutralize the signatures, the adversary has to succeed in two main hurdles. First, the adversary must have to find those computation(s) that only changes those *events* that are unrelated to the PoW algorithm. Second, the adversary must have to run these computation(s) in parallel to the PoW algorithm, which would again hamper the hashing rate, and thus the profit. In this work, we make a practical assumption that the attacker wants to maximize the profit and does not want to lose the computation cycles (hashing rate).

3.2 Data Collection

To better explain our work, we first describe what data we collect and how we collect it. We used the *perf* [5] tool to profile the processor’s *events* using HPC. In particular, we focus on hardware⁶ *events* (e.g., branch-misses), software⁷ *events* (e.g., page-faults), and hardware cache⁸ *events* (e.g., cache-misses) on CPU as the mining processes - depending on their design - require different type of resources. We profiled each program of both positive (mining) and negative (non-mining) class individually and collected a total 50 samples per program. Each sample consists of recordings of 28 *events* (described in Table 1) for 30s with a sampling rate 10 Hz, which means that each sample comprises 300 readings of 28 *events*, i.e., 8400 readings. To obtain clean signatures: (1) we profiled each program in its stable stage, i.e., omitting the bootstrapping phase; and (2) restarted the system to remove any trace of the previous sample.

For the positive class, we profiled a total of 11 cryptocurrencies discussed in Sect. 3.3. As the representatives of negative class, we chose: 3D rendering; 7z archive extraction of *tar.gz* files; H.264 video encoding of raw video; solving *mqueens* problem; Nanoscale Molecular Dynamics (NAMD) simulation; *Netflix* movie playback; execution of Random Forest (RF) machine learning algorithm; *Skype* video calls; *stress-ng* [6] stress test with CPU, memory, I/O, and disk workers together; playing *Team Fortress 2* game; and Visual Molecular Dynamics (VMD) modeling and visualization. It is worth mentioning that these user-tasks represent medium to high resource-intensive tasks.

We used two different systems to build our dataset for the experiments. The configuration of these systems are as follows: (1) *S1*, a laptop with an Intel Core i7-7500U @ 2.70 GHz (1 socket \times 2 cores \times 2 threads = 4 logical compute

⁶ Basic events, measured by Performance Monitoring Units (PMU).

⁷ Measurable by kernel counters.

⁸ Data- and instruction-cache hardware events.

Table 1. The *events* that we monitor using HPC. Here, HW = hardware, SW = software, and HC = hardware cache *event*.

Event	Type	Description	Event	Type	Description
branch-instructions	HW	N. of retired branch instructions.	iTLB-load-misses	HC	N. of instruction fetches that missed instruction TLB.
branch-load-misses	HW	N. of Branch load misses.	iTLB-loads	HC	N. of instruction fetches that queried instruction TLB.
branch-loads	HW	N. of Branch load accesses.	L1-dcache-load-misses	HC	N. of load misses at L1 data cache.
branch-misses	HW	N. of mispredicted branch instructions.	L1-dcache-loads	HC	N. of loads at L1 data cache.
bus-cycles	HW	N. of bus cycles, which can be different from total cycles.	L1-dcache-stores	HC	N. of stores at L1 data cache.
cache-misses	HC	N. of cache misses.	LLC-load-misses	HC	N. of load misses at the last level cache.
cache-references	HC	N. of cache accesses.	LLC-loads	HC	N. of loads at the last level cache.
context-switches	SW	N. of context switches.	LLC-store-misses	HC	N. of store misses at the last level cache.
cpu-migrations	SW	N. of times the process has migrated.	LLC-stores	HC	N. of stores at the last level cache.
dTLB-load-misses	HC	N. of load misses at data TLB.	mem-loads	HC	N. of memory loads.
dTLB-loads	HC	N. of load hits at data TLB.	mem-stores	HC	N. of memory stores.
dTLB-store-misses	HC	N. of store misses at data TLB.	page-faults	SW	N. of page faults.
dTLB-stores	HC	N. of store hits at data TLB.	ref-cycles	HW	N. of total cycles; not affected by CPU frequency scaling.
instructions	HW	N. of retired instructions.	task-clock	SW	The clock count specific to the task that is running.

resources) processor, 8 GB memory, 512 GB SSD storage, NVIDIA GeForce 940MX 2 GB dedicated graphic card, Linux kernel 4.14 and (2) *S2*, a laptop with an Intel Core i7-8550U @ 1.80 GHz (1 socket \times 2 cores \times 4 threads = 8 logical compute resources) processor, 16 GB memory, 512 GB SSD storage, Linux kernel 4.14.

All miner programs and the *perf* tool were launched in *user-mode*. Even though we did not use any system-level privileges, we believe that using *root* permissions for defense against cryptojacking is reasonable. The *perf* tool allows us to create per-process profile using *PID*. It is worth emphasizing that even though the dataset has been accumulated in a controlled setup, our experiments (discussed in Sect. 4) well simulate real-world scenario, where samples are collected in the real-time.

3.3 Cryptocurrencies and Miners

The probability of solving the cryptographic puzzle during mining is directly proportional to the miner’s computational power/resources. Consequently, the miners pool their resources to combine their hashing power with an aim to consistently earn a portion of the block reward by solving blocks quickly. Typically, the mining pools are characterized by their hashing power. Table 2 shows the top-10 mining pools [21] and the cryptocurrencies mined by them. These ten

mining pools collectively constitutes the biggest share (84% during Q1 2019) of the cryptomining business.

Table 2. Cryptocurrencies mined by the top-10 mining pools

N.	Mining pool	Cryptocurrency															
		BCD	BCH	BTC	BTM	DASH	DCR	ETC	ETH	LTC	SBTC	SC	UBTC	XMC	XMR	XZC	ZEC
1	BTC.com	X	✓	✓	X	X	X	X	X	X	✓	X	✓	X	X	X	X
2	AntPool	X	✓	✓	✓	✓	X	✓	✓	✓	X	✓	X	✓	X	X	✓
3	ViaBTC	X	✓	✓	X	✓	X	✓	✓	✓	X	X	X	X	X	X	✓
4	SlushPool	X	X	✓	X	X	X	X	X	X	X	X	X	X	X	X	✓
5	F2Pool	X	X	✓	X	✓	✓	✓	✓	✓	X	✓	X	✓	✓	✓	✓
6	BTC.top	X	✓	✓	X	✓	X	X	X	✓	X	X	X	X	X	X	X
7	Bitclub.network	X	X	✓	X	✓	X	✓	✓	X	X	X	X	X	✓	X	✓
8	BTCC	✓	✓	✓	✓	X	X	X	X	✓	X	X	X	X	X	X	X
9	BitFury	X	X	✓	X	X	X	X	X	X	X	X	X	X	X	X	X
10	BW.com	X	X	✓	X	X	X	✓	✓	✓	X	✓	✓	X	X	X	X

We considered all the cryptocurrencies mentioned in the Table 2 in our experiments. We used open-source miner programs to mine these cryptocurrencies. Each miner program was configured to mine with public mining pools and to utilize all available the CPUs present on the system. At the time of our experiments, the miner program for SC was not able to mine using only the CPU. Hence, we excluded SC from our experiments. To compensate SC, we included QRK whose mining algorithm - in contrast to other cryptocurrencies - uses multiple hashing algorithms. Table 3 shows the mining algorithm of different cryptocurrencies and the CPU miners that we used.

Table 3. Mining algorithm and CPU miner for different cryptocurrencies

Cryptocurrency	Mining algorithm	CPU miner
Bitcoin Diamond (BCD)	X13	cpuminer-opt 3.8.8.1
Bitcoin Cash (BCH), Bitcoin (BTC), SuperBitcoin (SBTC), UnitedBitcoin (UBTC)	SHA-256	cpuminer-multi 1.3.4
Bytom (BTM)	Tensority	bytom-wallet-desktop 1.0.2
Dash (DASH)	X11	cpuminer-multi 1.3.4
Decred (DCR)	Blake256-r14	cpuminer-multi 1.3.4
Ethereum Classic (ETC), Ethereum (ETH)	Ethash (Modified Dagger-Hashimoto)	geth 1.7.3
Litecoin (LTC)	scrypt	cpuminer-multi 1.3.4
Quark (QRK)	BLAKE + Grøstl + Blue Midnight Wish + JH + Keccak (SHA-3) + Skein	cpuminer-multi 1.3.4
Siacoin (SC)	BLAKE2b	gominer 0.6
Monero-Classical (XMC), Monero (XMR)	CryptoNight	cpuminer-multi 1.3.4
Zcoin (XZC)	Lyra2z	cpuminer-opt 3.8.8.1
Zcash (ZEC)	Equihash	Nicehash nheqminer 0.3a

Since our approach focuses on the underlying core PoW algorithm, we considered one currency for every mining algorithm mentioned in Table 3 and excluded BCH, SBTC, UBTC, ETC, and XMC in our study. As the proof-of-concept implementation, we considered only CPU-based miner programs because each computer has at least one CPU, which cryptojackers can harness to mine.

3.4 Classifier Design

In this section, we elucidate the design of our classification methodology. Algorithm 1 describes the pipeline of our classifier. Our supervised classification algorithm begins with splitting the base-dataset of 1100 samples (2 classes \times 11 instances \times 50 samples) into 90–10% stratified train-test sets, denoted as *raw_train_set* and *raw_test_set*. Then, these subsets are processed as follows:

Algorithm 1. Pseudo code for our supervised classification.

```

1: for each run  $i$  from 1 to 10 do
2:   Create raw_train_set and raw_test_set by 90–10% stratified partitioning.
3:   Data preprocessing
   • Replace NaN values from raw_train_set and raw_test_set with arithmetic mean
     of the considered event.
4:   Feature engineering
   • train_set := Extract_feature(raw_train_set)
   • test_set := Extract_feature(raw_test_set)
5:   Feature scaling
   • scaler := StandardScaler()
   • scaler.fit(train_set) ▷ Fit scaler on train_set
   • scaler.transform(train_set)
   • scaler.transform(test_set)
6:   Feature selection
   • Compute features' importance with forests of trees on train_set and select the
     most relevant features.
7:   Training
   • Learn the model parameters for the given classifier (RF/SVM) on the training
     set using grid search with 5-fold stratified CV.
8:   Predict/classify the test_set.
9: end for

```

1. *Data preprocessing*: The first step of any machine learning-based classification is to process the raw datasets to fix any missing value. Since each event we monitor returns a numerical value, we replace the missing values, if any, with the arithmetic mean of the respective event.
2. *Feature engineering*: In this step, we obtain features that can be used to train a machine learning model for our prediction problem. Here, we compute 12 statistical functions (listed in Table 4) for every event. This step converts each sample consisting of 300 readings (rows) \times 28 *events* (columns) to a single

row of 336 ($28 \text{ events} \times 12 \text{ features}$) data-points. The features extracted in this phase, hereinafter referred to as *train_set* and *test_set*, are used for the subsequent stages.

Table 4. The statistical functions that we used for our feature engineering phase

0.2, 0.4, 0.6, and 0.8 quantile	1, 2, and 3 sigma	Skewness
Arithmetic and geometric mean	Kurtosis	Variance

3. *Feature scaling:* It is an essential step to eliminate the influence of large-valued features because features with larger magnitude can dominate the objective function, and thus, deterring an estimator to learn from other features correctly. Hence, we standardize features using standard scaler, which removes the mean and scale the features to unit variance.
4. *Feature selection:* In machine learning, feature selection or dimensionality reduction is the process of selecting a subset of relevant features that are used in model construction. It aims to improve estimators' accuracy as well as to boost their performance on high-dimensional datasets. To do so, we calculate the importance of features using *forests of trees* [22] and select the most relevant features.
5. *Training:* The training phase consists of learning the model parameters for the given classifier on the training set, i.e., *train_set*. Given the nature of the problem, we resort to supervised machine learning procedures. In particular, we employed two of the most successful machine learning methods for classification, namely Random Forest (RF) [34] and Support Vector Machine (SVM) [30].
For model selection, we use grid search with 5-fold Cross Validation (CV). The validated hyper-parameters for RF and SVM are shown in Table 10 and Table 11, respectively in Appendix A. We chose standard range of values for the hyper-parameters [35].
6. *Prediction:* Finally, prediction is made on *test_set*.

The process is repeated ten times for a given experiment and the final results are computed over these ten runs.

4 Evaluation

We thoroughly evaluated our approach by performing an exhaustive set of experiments. We performed the following six different experiments: (1) *binary* classification (Sect. 4.1); (2) *currency* classification (Sect. 4.2); (3) *nested* classification (Sect. 4.3); (4) *sample length* (Sect. 4.4); (5) *feature relevance* (Sect. 4.5); and (6) *unseen miner programs* (Sect. 4.6). Table 5 describes the sample distribution in our base-dataset for each system, i.e., *S1* and *S2*. Here, sub-classes

of the mining task refer to the cryptocurrencies (discussed in Sect. 3.3) while sub-classes of the non-mining task refer to the actual user-tasks that belong to the negative class (mentioned in Sect. 3.2). We use the entire base-dataset (1100 samples per system) for each experiment, unless otherwise stated in an experiment.

Table 5. Dataset: name of the task, sub-classes per task, samples per sub-class, and total samples per task for each system

Task	Sub-classes per task	Samples per sub-class	Total samples per task
Mining	11	50	550
Non-mining	11	50	550

We evaluated our classifier using standard classification metrics: Accuracy, Precision, Recall, and F_1 score. To increase the confidence in our results, we report the mean and the margin of error for the results with 95% confidence interval from ten runs of each experiment for each of the evaluation metric. We use (\cdot) to indicate the best result for the metric and report the results as *mean \pm margin of error*.

4.1 Binary Classification

Our main goal is to identify whether a given instance represents the mining task or not. Hence, in this experiment, the label of each sample was defined as the positive or negative class, accordingly. Table 6 presents the results of the *binary* classification using both RF and SVM.

Table 6. Results for binary classification

System	Method	Accuracy	Precision	Recall	F1
<i>S1</i>	RF	1.000 \pm 0.000 \cdot	1.000 \pm 0.000 \cdot	1.000 \pm 0.000 \cdot	1.000 \pm 0.000 \cdot
	SVM	0.999 \pm 0.002	0.999 \pm 0.002	0.999 \pm 0.002	0.999 \pm 0.002
<i>S2</i>	RF	0.999 \pm 0.002 \cdot	0.999 \pm 0.002 \cdot	0.999 \pm 0.002 \cdot	0.999 \pm 0.002 \cdot
	SVM	0.990 \pm 0.018	0.991 \pm 0.016	0.990 \pm 0.018	0.990 \pm 0.018

Both the RF and SVM yielded superior performance. However, RF performed better than SVM on both the systems; the possible reason for the difference in classifiers' performance is their underlying designs - RF and SVM characterize their decision boundaries differently and also handle the outliers present in the dataset differently. On another side, the minute variations in the performance of a given classifiers across *S1* and *S2* are natural and expected; mainly due to

distinct dataset and data stratification. For the sake of brevity, we report the results only for RF for the subsequent experiments. We also present the details of parameters selected by grid search in Appendix B.

4.2 Currency Classification

The aim of this experiment is to understand the difficulty level of classification among various cryptocurrencies. Therefore, the input dataset for this experiment contained instances only of the cryptocurrencies. Table 7 lists the results of the *currency* classification.

Table 7. Results for currency classification

System	Accuracy	Precision	Recall	F1
<i>S1</i>	0.987 ± 0.017	0.992 ± 0.011	0.988 ± 0.016	0.985 ± 0.020
<i>S2</i>	0.986 ± 0.018	0.981 ± 0.027	0.985 ± 0.018	0.982 ± 0.024

Figure 2 depicts the confusion matrices for the classification among various cryptocurrencies to provide a better perception of the results. Here, Fig. 2(a) and Fig. 2(b) correspond to *S1* and *S2*, respectively. The confusion matrices are drawn using the aggregate results from all the ten runs. *Currency* classification is a multi-class classification problem, and some cryptocurrencies were misclassified among each other (see Fig. 2). Hence, the results are slightly lower than that of the *binary* classification.

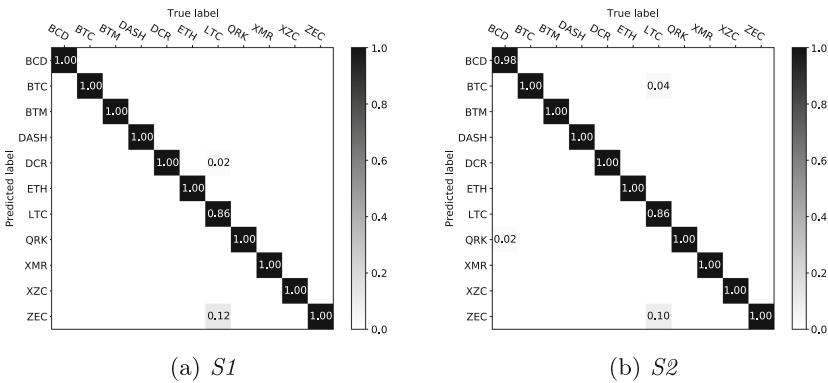


Fig. 2. Confusion matrix for classification among various cryptocurrencies

4.3 Nested Classification

This experiment represents a simulation of a real-world scenario. Here, we first classify whether a given instance belongs to the positive class. If so, we identify the cryptocurrency it belongs to. Essentially, *nested* classification is equivalent to performing *currency* classification on the instances classified as positive in the *binary* classification.

Table 8 shows the results of the *nested* classification. In the worst case, we expect the outcome of this experiment to be lower than that of the *binary* classification and *currency* classification together because a crucial aspect of such staged classification is that an error made in the prediction during the primary stage influences the subsequent stage; the results for *S1* shows this phenomenon. However, in a common scenario, the expected outcome of this experiment would be between the results for the *binary* classification and *currency* classification; the results for *S2* shows this effect.

Table 8. Results for nested classification

System	Accuracy	Precision	Recall	F1
<i>S1</i>	0.973 ± 0.020	0.972 ± 0.026	0.972 ± 0.020	0.967 ± 0.026
<i>S2</i>	0.996 ± 0.007	0.997 ± 0.006	0.996 ± 0.008	0.996 ± 0.008

4.4 Sample Length

The objective of this experiment is to understand the effect of length of the samples. For deployment in the real-world scenario, any solution - apart from being accurate - must be able to detect cryptojackers rapidly. To this end, we performed the *binary* classification of samples of a length of 5, 10, 15, 20, 25, and 30s, each in separate experiments. It is worth mentioning that we used samples of identical length for both the training and testing. Figure 3 shows the F_1 score when using samples of different length.

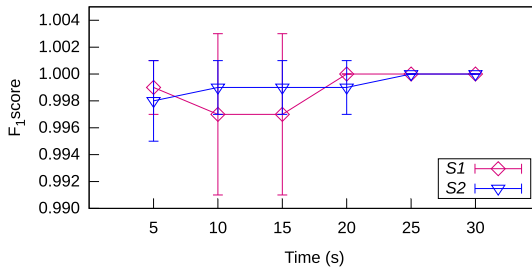


Fig. 3. F_1 score for different sample lengths (whiskers represent margin of error)

As explained in Sect. 3.1, the task of mining is to repeatedly execute the core PoW algorithm. Hence, even samples of shorter length can grasp the signature. As shown in Fig. 3, our system can achieve high performance with samples of 5s. The dip in the curve for *S1* corresponds to the thousandths digit of the F_1 score. For the sake of brevity, we omitted the results for sample shorter than five seconds and only focus on the required minimum sample length to attain high performance with our solution.

4.5 Feature Relevance

Next, we focus on our feature selection process (mentioned in Sect. 3.4). After calculating the importance of features, we sorted them in ascending order of their importance and selected the first- $\Psi\%$ features to do the *binary* classification. The key idea here is to identify the lower-limit of (even less important) features required to obtain the best performance. Figure 4 depicts the F_1 score when using first- $\Psi\%$ features.

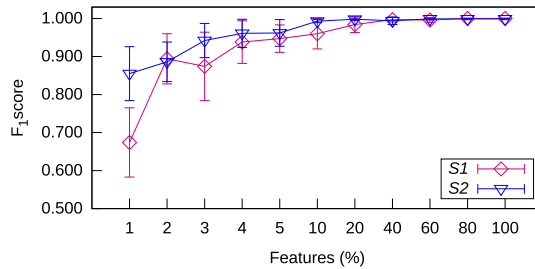


Fig. 4. F_1 score for first- $\Psi\%$ features (whiskers represent margin of error)

Since the features are sorted in the ascending order of their importance, we begin with the feature with lowest significance. Intuitively, including important features further improves the classification process. As shown in the Fig. 4, our classifier attains high performance on both the systems using only the first-40% (less relevant) features, which verifies/approves our feature engineering and selection process.

4.6 Unseen Miner Programs

There can be several different miner-programs available to mine a given cryptocurrency. These programs come from different developers/sources. Consequently, there can be some variations in the behavior of the miner-program itself, e.g., in the code section before/after the PoW function or handling (on the programming-side) a correct nonce found while mining. The reason is that they are developed by different developers, which intuitively will cause variations.

Training the model for each program may not be feasible for a variety of reasons. Hence, to investigate the effectiveness of our approach in such a situation, we set up this experiment. Here, we selected the *binary* classification as the target where the samples from all the mining and non-mining tasks were labeled as the positive or negative class, respectively. However, we chose two additional miner programs for BTC, namely, BFGMiner 5.5 and cgminer 4.10. We collected additional 50 samples each for BFGMiner 5.5 and cgminer 4.10 on both *S1* and *S2* separately. In the training phase, we used samples from one of the three miner programs for BTC. On the contrary, we used samples from one of the other two miner programs for BTC during the testing phase. Table 9 presents the results of classifying samples from the miner programs that were unseen in the training phase.

Table 9. Results for unseen miner programs

System	Task	Accuracy	Precision	Recall	F1
<i>S1</i>	α_β	0.997 ± 0.006	0.997 ± 0.006	0.997 ± 0.006	0.997 ± 0.006
	α_γ	0.998 ± 0.005	1.000 ± 0.000	0.997 ± 0.006	0.998 ± 0.004
	β_α	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
	β_γ	0.999 ± 0.001	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
	γ_α	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
	γ_β	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
<i>S2</i>	α_β	0.999 ± 0.001	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
	α_γ	0.998 ± 0.002	0.997 ± 0.003	0.997 ± 0.003	0.997 ± 0.003
	β_α	0.999 ± 0.002	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003
	β_γ	0.999 ± 0.001	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
	γ_α	0.999 ± 0.001	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
	γ_β	0.999 ± 0.001	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002

The notation X_Y means that the training was done with the samples from X while the testing was done with the sample from Y for BTC. Here, $\alpha =$ cpuminer-multi 1.3.4, $\beta =$ BFGMiner 5.5, $\gamma =$ cgminer 4.10. It is important to mention that these results are for the classification of all the mining and non-mining tasks with BTC being trained and tested upon samples from different programs. As discussed in Sect. 3.1, the miners have to execute the same core PoW algorithm for a given cryptocurrency. Hence, samples from different miner programs for a cryptocurrency retain the same signatures, which is reflected in our results.

Cross-Platform Classification: Next, we evaluate the transferability of the profiles built by our approach. We perform binary classification with additional samples from $S1'$ (a system with the same processor as $S1$) and $S2'$ (a system with the same processor as $S2$), and found that: (1) the profile of an algorithm on a given processor can be used with the help of machine learning technique to classify samples from another system with the same processor and (2) on the contrary, the profile of an algorithm on one processor is not useful to perform classification of samples from another processor.

5 Limitations

In this section, we address the potential limitation of our proposed approach.

5.1 Zero-Day Cryptocurrencies

A zero-day cryptocurrency would be a currency that uses a completely new or custom PoW algorithm that was never seen before. As a matter of fact, for a cryptocurrency to obtain market value: (1) its core-network should be supported by miners/pools; and (2) its PoW algorithm must be accepted by the crypto-community and tested mathematically for its robustness. Therefore, the PoW algorithm for a new cryptocurrency would become public by the time it gets ready for mining, which would give us sufficient time to capture this new cryptocurrency's signature and to train our model.

Importantly, miners prefer to mine cryptocurrencies that are more profitable and avoid hashing the less rewarding ones. As it happens to be, more profitable cryptocurrencies are indeed popular and their PoW algorithms are certainly known to the public. In our experiments, we considered all the popular cryptocurrencies, and our results (presented in Sect. 4) demonstrate the high quality of our proposed approach along various dimensions.

5.2 Scalability

The key concept of our approach is to profile the behavior of a processor's *events* for mining algorithms. Since there are only a finite number of CPUs/GPUs, procuring their signature is only a matter of data collection. It might appear as a ponderous job and may be seen as a limitation of our work. But, once it is accomplished for the available CPUs/GPUs, maintaining it is relatively simpler as merely a limited number of CPUs/GPUs are released over a period of time.

5.3 Process Selection

As mentioned in Sect. 3.2, our system requires per program/process-based recording of HPC for different *events* as the input to the classifier. In practice, several processes run in the system. Hence, monitoring each process may consume time and can be seen as a limitation of our work. However, as shown

in Fig. 3, our system can achieve high performance even with samples of 5s. On another side, the miner programs attempt to use all the available resources. Therefore, an initial sorting/filtering of processes based on their resource usage can help to boost the detection process in real-time.

5.4 Restricted Mining

A mining strategy to evade detection from our proposed methodology can be *restricted mining* that aims to change the footprint of the mining process. The essence here is that the miner program/process can be modified to perform arbitrary operations during mining. But, such maneuvers would directly affect the hashing rate and consequently the profits of mining; making the task of mining less appealing. Nevertheless, like any signature-based detection technique, it may be seen as a limitation of our work.

6 Conclusion and Future Works

Cybercriminals have developed several proficient ways to exploit cryptocurrencies with an aim to commit many unconventional financial frauds. Covert cryptomining is one of the most recent means to monetize the computational power of the victims. In this paper, we present our efficient methodology to identify covert cryptomining on users' machine. Our solution has a broader scope - compared to the solution that are tailored to a particular cryptocurrency or a specific form (e.g., browser-based) of cryptomining on computers - as it targets the core PoW algorithms and uses the low-performance overhead HPC that are present in modern processors to create discernible signatures. We tested our generic approach against a set of rigorous experiments that include eleven distinct cryptocurrencies. We found that our classifier attains high performance even with short samples of five seconds.

We believe that our approach is valid to distinguish GPU-based miners because dedicated profiling tools, such as the *nvprof* [4] tool for NVIDIA GPUs, allow us to monitor GPU *events*. Apart from most of the standard *events* found on CPUs, GPUs have several dedicated *events* that can assist in creating unique signatures for GPUs. Nevertheless, we keep such investigation as part of our future work. We will also perform our experiments with a larger set of systems (CPUs) to observe the generalization of our approach. We also hope to release a desktop application for run-time identification of covert cryptomining.

Appendix A Validated Hyper-parameters

The validated hyper-parameters for RF and SVM are shown in Table 10 and Table 11, respectively.

Table 10. Hyper-parameters validated for RF classifier

Parameter	Validated values	Effect on the model
<i>n_estimators</i>	{10, 25, 50, 75, 100, 125, 150}	Number of trees use in the ensemble
<i>max_depth</i>	[2, ∞)	Maximum depth of the trees
<i>max_features</i>	'auto', 'log2'	Number of features to consider when looking for the best split
<i>split_criterion</i>	<i>gini, entropy</i>	Criterion used to split a node in a decision tree
<i>bootstrap</i>	<i>true, false</i>	Bootstrap Aggregation (a.k.a. bagging) is a technique that reduces model variances (overfitting) and improves the outcome of learning on limited sample or unstable datasets
<i>random_state</i>	10	The seed used by the random number generator

Table 11. Hyper-parameters validated for SVM classifier

Parameter	Validated values	Effect on the model
<i>kernel</i>	'rbf', 'poly', 'sigmoid'	Specifies the kernel type to be used in the algorithm
<i>C</i>	[10^{-3} , 10^5]	Regularization parameter that controls the trade-off between the achieving a low training error and a low testing error that is the ability to generalize your classifier to unseen data
γ	'auto', [10^{-7} , 10^3]	Shape parameter of the RBF kernel which defines how an example influence in the final classification
<i>degree</i>	default=3	Degree of the polynomial kernel function ('poly'). Ignored by all other kernels
<i>random_state</i>	10	The seed of the pseudo random number generator used when shuffling the data for probability estimates

Appendix B Parameters selected by grid search

Here, we list the frequency of parameter-values selected by grid search over ten-runs of different experiments. Table 12 corresponds to *binary* classification experiment with SVM while Table 13 corresponds to *binary*, *currency*, and *full* classification experiments with RF for both *S1* and *S2*.

Table 12. *Binary* classification with SVM

Parameter	Value	N. of times selected on <i>S1</i>	N. of times selected on <i>S2</i>
<i>kernel</i>	'rbf'	7	6
	'poly'	1	0
	'sigmoid'	2	4
<i>C</i>	0.01	1	0
	0.1	0	4
	1	1	1
	10	3	2
	100	2	2
	1000	3	1
γ	0.0001	2	1
	0.001	1	4
	0.01	2	1
	0.1	2	0
	'auto'	3	4

Table 13. Different classifications with RF

Parameter	Value	Binary classification		Currency classification		Full classification	
		N. of times selected on <i>S1</i>	N. of times selected on <i>S2</i>	N. of times selected on <i>S1</i>	N. of times selected on <i>S2</i>	N. of times selected on <i>S1</i>	N. of times selected on <i>S2</i>
<i>bootstrap</i>	<i>true</i>	10	10	10	10	10	10
	<i>false</i>	0	0	0	0	0	0
<i>max_features</i>	'log2'	3	4	5	3	5	1
	'auto'	7	6	5	7	5	9
<i>max_depth</i>	2	0	0	4	1	0	0
	3	5	5	5	5	5	1
	4	2	1	0	3	4	7
	5	2	2	1	1	1	2
	6	1	0	0	0	0	0
	7	0	2	0	0	0	0
<i>split_criterion</i>	<i>gini</i>	9	9	10	6	10	10
	<i>entropy</i>	1	1	0	4	0	0
<i>n_estimators</i>	10	2	3	0	5	0	0
	25	5	1	1	2	1	0
	50	2	1	4	1	0	1
	75	0	0	2	2	0	0
	100	0	0	2	0	5	5
	125	1	4	0	0	3	1
	150	0	1	1	0	1	3

References

1. Coinhive. <https://tinyurl.com/ybsy89k2>
2. CoinMarketCap. <https://tinyurl.com/o94fhlw>
3. Crypto-Loot. <https://tinyurl.com/y76ppd5g>
4. The *nvprof* Tool. <https://tinyurl.com/y8tqxn74>
5. The *perf* Tool. <https://tinyurl.com/ybpmxw8>
6. The *stress-ng* Tool. <https://tinyurl.com/my6ehnj>
7. An Italian Bank's Server was Hijacked to Mine Bitcoin (2017). <https://tinyurl.com/yac8c8jq>
8. Persistent Drive-by Cryptomining Coming to a Browser Near You (2017). <https://tinyurl.com/yd5roadb>
9. Bitcoin Energy Consumption Index (2018). <https://tinyurl.com/y8w5yj9l>
10. Cryptojacking: A Modern Cash Cow (2018). <https://tinyurl.com/y28eqdav>
11. Cryptojacking Attack Found on Los Angeles Times Website (2018). <https://tinyurl.com/y8ghcvmd>
12. FacexWorm Targets Cryptocurrency Trading Platforms, Abuses Facebook Messenger for Propagation (2018). <https://tinyurl.com/yd2zja9q>
13. Greedy Cybercriminals Host Malware on GitHub (2018). <https://tinyurl.com/y9qon8ch>
14. Is Bitcoin Mining Profitable or Worth it in 2018? (2018). <https://tinyurl.com/ybnydb8g>
15. KSN Report: Ransomware and Malicious Cryptominers 2016–2018 (2018). <https://tinyurl.com/y29kybtx>
16. Revenues Down, Hashrates Up: 2018 Mining Outlook by the Numbers (2018). <https://tinyurl.com/yc586s9v>
17. rTorrent Client Exploited in the Wild to Deploy Monero Crypto-miner (2018). <https://tinyurl.com/yaqy7u3k>
18. Tesla Hackers Hijacked Amazon Cloud Account to Mine Cryptocurrency (2018). <https://tinyurl.com/y9epv3do>
19. UK ICO, USCourts.gov. Thousands of Websites Hijacked by Hidden Cryptomining Code after Popular Plug-in Pwned (2018). <https://tinyurl.com/y7upaxgv>
20. WebCobra Malware Uses Victims' Computers to Mine Cryptocurrency (2018). <https://tinyurl.com/ycuhowb3>

21. Bitcoin Mining Pools (2019). <https://tinyurl.com/y8pdk922>
22. Feature Importances with Forests of Trees (2019). <https://tinyurl.com/y3nld2h>
23. IBM X-Force Threat Intelligence Index (2019). <https://tinyurl.com/y5nbpvrve>
24. SonicWall Cyber Threat Report (2019). <https://tinyurl.com/y3wj69s7>
25. Under the Hood of Cyber Crime (2019). <https://tinyurl.com/ydhauj8x>
26. Bonneau, J., et al.: SoK: research perspectives and challenges for bitcoin and cryptocurrencies. In: 36th IEEE S&P, pp. 104–121 (2015)
27. Chiappetta, M., et al.: Real-time detection of cache-based side-channel attacks using hardware performance counters. *Appl. Soft Comput.* **49**, 1162–1174 (2016)
28. Comodo Cybersecurity: Global Threat Report Q2 2018 Edition (2018). <https://tinyurl.com/y8eos9pl>
29. Conti, M., et al.: On the economic significance of ransomware campaigns: a bitcoin transactions perspective. *Comput. Secur.* **79**, 162–189 (2018)
30. Cortes, C., Vapnik, V.: Support vector networks. *Mach. Learn.* **20**(3), 273–297 (1995)
31. Cyber Threat Alliance (CTA): The illicit cryptocurrency mining threat report (2018). <https://tinyurl.com/yco7cykl>
32. Demme, J., et al.: On the feasibility of online malware detection with performance counters. In: 40th ISCA, pp. 559–570 (2013)
33. Gangwal, A., Conti, M.: Cryptomining cannot change its spots: detecting covert cryptomining using magnetic side-channel. *IEEE Trans. Inf. Forensics Secur.* **15**(1), 1630–1639 (2019)
34. Ho, T.K.: Random decision forests. In: 3rd ICDAR, pp. 278–282 (1995)
35. Hsu, C.W., et al.: A practical guide to support vector classification. Tech. rep. (2003)
36. Huang, D.Y., et al.: Botcoin: monetizing stolen cycles. In: 21st NDSS, pp. 1–16 (2014)
37. Konoth, R.K., et al.: MineSweeper: an in-depth look into drive-by cryptocurrency mining and its defense. In: 25th ACM CCS (2018)
38. Liu, J., et al.: A novel approach for detecting browser-based silent miner. In: 3rd IEEE DSC, pp. 490–497 (2018)
39. Mora, C., et al.: Bitcoin emissions alone could push global warming above 2 °C. *Nat. Clim. Change* **8**(11), 931–933 (2018)
40. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). <https://tinyurl.com/3f4a6lr>
41. Rauchberger, J., et al.: The other side of the coin: a framework for detecting and analyzing web-based cryptocurrency mining campaigns. In: 13th ARES, pp. 1–10 (2018)
42. R uth, J., et al.: Digging into browser-based crypto mining. arXiv preprint: 1808.00811 (2018)
43. Tahir, R., et al.: Mining on someone else’s dime: mitigating covert mining operations in clouds and enterprises. In: Dacier, M., Bailey, M., Polychronakis, M., Antonakakis, M. (eds.) RAID 2017. LNCS, vol. 10453, pp. 287–310. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66332-6_13
44. Wang, W., Ferrell, B., Xu, X., Hamlen, K.W., Hao, S.: SEISMIC: SEcure In-lined Script Monitors for Interrupting Cryptojacks. In: Lopez, J., Zhou, J., Soriano, M. (eds.) ESORICS 2018. LNCS, vol. 11099, pp. 122–142. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98989-1_7
45. Wang, X., et al.: ConFirm: detecting firmware modifications in embedded systems using hardware performance counters. In: 34th IEEE/ACM ICCAD, pp. 544–551 (2015)

46. Wang, X., et al.: Hardware performance counter-based malware identification and detection with adaptive compressive sensing. *ACM TACO* **13**(1), 1–23 (2016)
47. Wang, X., Karri, R.: NumChecker: detecting kernel control-flow modifying rootkits by using hardware performance counters. In: 50th DAC, pp. 1–7 (2013)
48. Yuan, L., et al.: Security breaches as PMU deviation: detecting and identifying security attacks using performance counters. In: 2nd ACM SIGOPS APSys, pp. 1–6 (2011)



Lightweight Virtual Payment Channels

Maxim Jourenko¹(✉), Mario Larangeira^{1,2}, and Keisuke Tanaka¹

¹ Department of Mathematical and Computing Sciences, School of Computing,
Tokyo Institute of Technology, Tokyo-to Meguro-ku Ookayama 2-12-1,
Meguro City W8-55, Japan

jourenko.m.ab@m.titech.ac.jp, mario@c.titech.ac.jp,
keisuke@is.titech.ac.jp

² Input Output Hong Kong, Hong Kong, People's Republic of China
mario.larangeira@iohk.io
<http://iohk.io>

Abstract. Blockchain systems have severe scalability limitations e.g., long confirmation delays. Layer-2 protocols are designed to address such limitations. The most prominent class of such protocols are payment channel networks e.g., the Lightning Network for Bitcoin where pairs of participants create channels that can be concatenated into networks. These allow payments across the network without interaction with the blockchain. A drawback is that all intermediary nodes within a payment path must be online. Virtual Channels, as recently proposed by Dziembowski et al. (CCS'18), allow payments without this limitation. However, these can only be implemented on blockchains with smart contract capability therefore limiting its applicability. Our work proposes the notion of *Lightweight Virtual Payment Channels*, i.e. only requiring timelocks and multisignatures, enabling Virtual Channels on a larger range of blockchain systems of which a prime example is Bitcoin. More concretely, other contributions of this work are (1) to introduce a fully-fledged formalization of our construction, and (2) to present a simulation based proof of security in Canetti's UC Framework.

Keywords: Blockchain · Payment channels · Cryptocurrency

1 Introduction

Blockchains implement decentralized ledgers via consensus protocols run by mutually distrustful parties. Despite the novelty of such design, it has inherent limitations, for example, effectively all transactions committed to the ledger have to be validated by all parties. Croman et al. [6] showed that this severely limits a blockchain's throughput. Moreover, there is a minimal delay between

This work was supported by the Input Output Cryptocurrency Collaborative Research Chair funded by IOHK, JST CREST JPMJCR14D6, JST OPERA, JSPS KAKENHI 16H01705, 17H01695.

submission of a transaction and verification thereof that is intrinsic to the system's security, e.g. one hour in the case of Bitcoin.

Layer-2 protocols, such as payment channel networks, allow confirmation of transactions outside the consensus protocol while using it as fallback. These protocols are referred to as “off-chain” protocols in contrast to processing transactions via the consensus protocol “on-chain”. An elementary protocol realizes *channels* and commonly works as follows: Two (or more) parties put together their funds and lock them on-chain by requiring a 2-out-of-2 (n-out-of-n) multisignature to claim them. Then these funds are spent by another transaction or a tree of transactions. These transactions represent the distribution of funds between both parties and are not committed to the blockchain except when parties enforce the fund distribution on-chain and unlock the funds. The parties can perform a payment, i.e. update the balance distribution within the channel, by recomputing that tree of transactions while invalidating previous transaction trees. Payments between parties are processed immediately and only involve interaction between the two parties. Channels can be extended to form channel networks by using Hashed Time Lock Contracts (HTLC) [7, 15]. Payments are performed by finding a path from payer to payee within the network and atomically replicating the payment on each channel along that path. A drawback of HTLCs is that a payment requires interaction with all intermediary nodes within a path. Virtual State Channels as proposed by Dziembowski et al. [8, 9] devise a technique for creation of channels that allow execution of state machines instead of being limited to payments, and use an off-chain protocol that expands the network with new channels. The latter reduces the network's diameter yielding shorter payment paths, and allowing parties to perform payments without interacting with any intermediary nodes if they are adjacent in the now extended network. However, this construction requires blockchain with smart-contract capability, therefore not applicable to Bitcoin. Later we will see that this work addresses this limitation with a novel construction.

Use cases for virtual channels are manifold. A virtual payment channel provides the same benefits to the two parties sharing one as pairwise payment channels without the need to set it up by committing transactions to the ledger that can incur expensive fees. Payments can be executed offchain, without interaction with a third party and without incurring any fees, e.g. for routing an HTLC, making rapid micro-payments viable. This could enable new services such as a service-gateway. Such a gateway would consist of a node that sets up payment channels with different service provider that operate using micro-transactions, e.g. Video on Demand (VoD) services that bill by watch-time. A user could then create one payment channel with the gateway node and with the use of virtual channels created ad-hoc connections to the different (VoD) services instead of having to set up individual payment channels with each service they want to use. A more general use case is that virtual channels allow payment hubs, that have a high degree within a payment channel network, to interconnect their individual partners in exchange for a fee.

Related Work. HTLCs allow atomic payments across multiple hops. This is done by performing a conditional payment in each channel along a path from payer to payee. Executing a payment requires revealing a secret $x \in \mathbb{N}$ such that $H(x) = y$ where H is a cryptographic hash function. After setup, starting from the payee each node within the payment path reveals x to its predecessor. This proves that the payment can be enforced on-chain which allows parties to resolve the payment by performing it off-chain. A timelock is used to cancel the transaction after a preset amount of time which unlocks the funds from the conditional payment. Although our construction can be used to enable payments across a payment channel network by creating a virtual channel between payer and payee, we argue that our work is orthogonal to HTLCs and both techniques can be used in tandem. First our construction is used to expand the underlying payment channel network with additional virtual channels and then HTLCs can be used to perform payments across this expanded infrastructure.

Dziembowski et al. introduced Virtual State Channels [8] and State Channel Networks [9]. A *state channel* depends on a smart contract previously committed to the blockchain. It contains (1) application specific code, and (2) code for state channel management. More specifically parties can send messages to the smart contract changing its state according to (1), or compute a state-transition message where the resulting state is computed by the parties and summarized in the state-transition message for (2). The state-transition message can be kept off-chain, and only committed to the blockchain in case of parties' dispute. A virtual state channel can be built on top of two channels that were previously created in this manner. Similar to our work, virtual channels cannot be open indefinitely but have a *fixed* lifetime that is decided upon construction. In contrast to our work this technique requires a blockchain with smart-contract capability. Chakravarty et al. proposed Enhanced Unspent Transaction Outputs (EUTxO) [4] and constructed the Hydra Protocol [5]. EUTxO enables running constraint emitting state machines on top of a ledger which is used to setup a Hydra heads among a set of parties. This allows them to take their funds off-chain and confirm transactions with these funds among the participants of the Hydra head. Although parties can interact with each other using arbitrary transactions as they would on-chain, no new participants can be added to the Hydra head which is in contrast to payment channel networks. Moreover implementing Hydra requires blockchains with EUTxO capability limiting its applicability.

Our Contributions. This work proposes a new variant of Virtual Channels, we name it *Lightweight Virtual Payment Channels*, that is based on UTXO and requires only multisignatures and timelocks, that is, it does not require smart-contracts, yielding the first virtual channel construction implementable on blockchains such as Bitcoin, which currently has the highest market capitalization of all cryptocurrencies¹ and still is the most widely used, and blockchains operating with the recently introduced EUTxO [4] effectively improving the state of the art in both cases.

¹ <https://coinmarketcap.com>.

In a nutshell, our Layer-2 protocol for Virtual Payment Channels takes two payment channels between three parties as input, and opens three payment channels, i.e. one for each pair of parties. Our protocol can be applied iteratively allowing for virtual payment channels across multiple hops of the underlying payment channel network. Our construction (1) can be used to expand a payment channel network with virtual payment channels, (2) allows payments without interaction with intermediary nodes if payer and payee share a virtual payment channel, (3) can be used in tandem with HTLCs and (4) can be used with different payment channel implementations as Duplex Payment Channel [7], Lightning [15], Eltoo [14]. We formalize our work in Canetti’s Universal Composability (UC) Framework [1] by introducing a functionality for lightweight virtual payment channels $\mathcal{F}_{\text{LVPC}, \mathcal{F}_{\text{PWCH}}}$. Although formalizations for ledgers, including Bitcoin, within the UC framework exist [8,9] we present the first global functionality $\mathcal{G}_{\text{UTXO-Ledger}}$ for an Unspent Transaction Output (UTXO) based ledger. Moreover we present an auxiliary functionality $\mathcal{F}_{\text{Script}}$ modeling a scripting language modelling access to *timelocks* and *multisignatures*. Our construction makes use of $\mathcal{G}_{\text{CLOCK}}$ by Katz et al. [10], modified by Kiayias et al. [11,12] and \mathcal{F}_{SIG} by Canetti et al. [2]. We present pseudo-code protocols `Open_VC`, `Close_VC` and `Enforce_VC`.

Structure of This Work. We briefly introduce notation and the model used in this work in Sect. 2. Next we formalize a UTXO based ledger and their components in Sect. 3. Afterwards we give a high-level description and analysis of our approach in Sect. 4 before presenting protocols in Sect. 5. We formalize our approach in the UC Framework in Sect. 6. Lastly we discuss directions for future work in Sect. 7.

2 Preliminaries

Let $\text{negl}(n)$ denote the negligible function. Furthermore consider the standard definition for *computational indistinguishability* $X \approx_c Y$, i.e., there is no PPT algorithm D such that D can distinguish between two ensembles of probabilistic distributions $X = \{X_n\}_{n \in \mathbb{N}}$ and $Y = \{Y_n\}_{n \in \mathbb{N}}$, in other words $\Pr[D(X_n, 1^n) = 1] - \Pr[D(Y_n, 1^n) = 1] \leq \text{negl}(n)$. Moreover let \cup , \cap and \setminus denote set union, intersection, subtraction, and \emptyset be the empty set. We make frequent use of tuples to structure data. Assume a tuple of type \mathcal{A} is defined as (a_0, a_1, \dots, a_n) and A is an instantiation of such a tuple. For simplicity we denote the entry labeled a_i of A as $A.a_i$.

The Adversarial and Computational Model. We model the execution of our protocol π via the Universal Composability (UC) Framework with Global Setup by Canetti et al. [3] where all the entities are PPT Interactive Turing Machines (ITM), and the global setup is given by the global functionality \mathcal{G} , and the execution is controlled by the environment \mathcal{Z} . In this simulation based model, all parties from π have access to the auxiliary functionality \mathcal{F}_{aux} , i.e., $\pi^{\mathcal{F}_{\text{aux}}}$, in the hybrid world execution $\text{HYBRID}_{\pi^{\mathcal{F}_{\text{aux}}}, \mathcal{A}, \mathcal{Z}}$ in the presence of the

adversary \mathcal{A} which can see and delay the messages within a communication round. Whereas the ideal execution, *i.e.*, $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, is composed by the functionality \mathcal{F} in the presence of the simulator \mathcal{S} . In both executions, the environment \mathcal{Z} access the global functionality \mathcal{G} . We assume static corruption by a malicious adversary. Given the randomness r and input z , the environment \mathcal{Z} drives both executions $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\text{HYBRID}_{\pi^{\mathcal{F}_{aux}},\mathcal{A},\mathcal{Z}}$, and output either 1 or 0. Therefore, let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\text{HYBRID}_{\pi^{\mathcal{F}_{aux}},\mathcal{A},\mathcal{Z}}$ be respectively the ensembles $\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n, z, r)\}_{n \in \mathbb{N}, z \in \{0,1\}^*}$ and $\{\text{HYBRID}_{\pi^{\mathcal{F}_{aux}},\mathcal{A},\mathcal{Z}}(n, z, r)\}_{n \in \mathbb{N}, z \in \{0,1\}^*}$ of the outputs of \mathcal{Z} for both executions. Thus, we say that $\pi^{\mathcal{F}_{aux}}$ *realizes* \mathcal{F} in the \mathcal{F}_{aux} -Hybrid model when, there exist a PPT simulator \mathcal{S} , such that for all PPT \mathcal{Z} , we have $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx_c \text{HYBRID}_{\pi^{\mathcal{F}_{aux}},\mathcal{A},\mathcal{Z}}$.

Communication Model. We assume synchronous communication where time is split into communication rounds. If any party sends a message to a receiving party within a round, the message reaches the receiving party at the beginning of the following communication round.

3 The UTXO Model

In the following we review the notion of Unspent Transaction Outputs (UTXO), UTXO based ledger and transactions. Thereafter we briefly review payment channel.

Overview. A UTXO wraps an amount of currency and comes with a script. To claim an UTXO, a witness needs to be provided s.t. if provided as input into the script, it evaluates to *true*. A UTXO based ledger's state is a set of all UTXO that are in circulation. The state can be altered using transactions that contain a set of inputs and a list of outputs. Each input references a UTXO and contains its witness. Each output is a newly defined UTXO. Submitting such a transaction to the ledger alters its state by removing the UTXO referenced in the inputs and adding the UTXO defined in the outputs. Moreover, a transaction might contain a point in time $t \in \mathbb{N}$ called *timelock* s.t. it is not possible to submit the transaction to the ledger before time t . Note that in this work we only make use of scripts that verify *multisignatures*. More formally, we have the following.

The UTXO Tuples. The UTXO are tuples (b, Party) , where $b \in \mathbb{N}$ is the amount of coins and Party is a set of parties. We denote a reference to a UTXO out by $\text{ref}(\text{out})$. Note UTXO are uniquely identifiable, e.g. in Bitcoin UTXOs are identifiable by the hash of the transaction in which they were defined, and their index within the transaction's outputs.

Funding UTXO Pattern. A Funding UTXO $\text{F_UTXO}(x, \mathcal{P}_0, \mathcal{P}_1)$ is of the form $(x, \{\mathcal{P}_0, \mathcal{P}_1\})$ where $x \in \mathbb{N}$ and $\mathcal{P}_0, \mathcal{P}_1$ are parties.

Transactions. A transaction is a tuple $(t, \text{In}, \text{Out})$ where $t \in \mathbb{N}$ is a point in time specifying a *timelock*, Out is a list of UTXO and In is a set of inputs. An input is of the form (ref, Σ) where ref is a reference to an UTXO and Σ is a set

of signatures. A transaction is valid and can be committed to the ledger after time t , if all UTXO in Tr.Ref are unique, each input contains a correct witness and it holds that $\sum_{\text{ref}(i) \in \text{Tr.Ref}} i.b \geq \sum_{o \in \text{Tr.Out}} o.b$, i.e. it spends at most as many funds as it claims.

UTXO Ledger. A UTXO ledger's state is represented by a set of UTXO U . Parties may read the ledger's state and change it by submitting a valid transaction. All UTXO referenced by the inputs are removed from U and all UTXO in the outputs are added to the ledger. As conventionally done in the literature, in the remainder of this work we assume that any such transaction will be processed on the ledger within duration $\Delta \in \mathbb{N}$.

Transactions as Graphs. Transactions submitted to alter a ledger's state form a tree where transactions themselves form nodes, the UTXO specified in their outputs form outgoing edges and UTXO referenced in their inputs form incoming edges. Note that transactions within a tree can only be committed to the ledger if its root is committed to the ledger.

Partial Mappings. We abstract away from transactions and represent them as partial mappings of UTXO of the form (In, Out) where In, Out are UTXO that represent the transaction's inputs and outputs respectively. We assume there is a function ϕ that takes a mapping (In, Out) and time t and outputs a respective transaction with timelock t . Analogously ϕ^{-1} is a function that takes a transaction and outputs a mapping and timelock.

Pairwise Payment Channel. A pairwise payment channel allows two parties to exchange funds without committing a transaction to the ledger for the individual payments. Such a channel is setup by having parties commit a transaction on the ledger that collects some of each party's UTXO and spends all of it within a Funding UTXO. Committing this transaction on the ledger locks these funds. The Funding UTXO is spent by a transaction subtree representing the channel's state where committing it to the ledger unlocks and returns all of the parties' funds, however, instead, the parties hold off committing them. When executing a payment, they update the transaction subtree to represent the new state while invalidating the previous subtree. Invalidation can be done by spending the Funding UTXO with a transaction that has a timelock of at least Δ less than the previous subtree. We remark that alternative invalidation methods do exist [7, 14, 15]. The channel is closed by committing the transaction subtree or a transaction summarizing it onto the ledger.

4 Overview of the Construction

The construction consists of three protocols, `Open_VC` (Fig. 3), `Close_VC` (Fig. 4) and `Enforce_VC` (Fig. 5) used for setup, tear-down and dispute of virtual channels respectively. We remark that the executions of `Open_VC` and `Close_VC` require consent between all involved parties, and `Enforce_VC` can be executed by a party unilaterally.

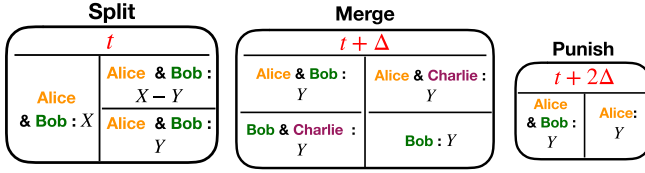


Fig. 1. Illustrations of transactions used through out this work represented as nodes of a transaction graph. A transaction’s inputs are listed on the left-hand-side whereas a transaction’s outputs are on the right-hand-side. The value on top represents the transaction’s timelock.

Types of Transactions. We use three types of transactions, they are are *Split*, *Merge* and *Punish* transactions as illustrated in Fig. 1. A Split transaction spends a Funding UTXO and creates two new Funding UTXO between the same pair of parties. A Merge transaction takes two Funding UTXO between three parties and equal balance as input, and creates two UTXO with the same amount of funds as the inputs each. One is a Funding UTXO between the two parties that do not share a Funding UTXO within the inputs, and one is a UTXO that gives funds to the third party. Lastly the Punish transaction takes a Funding UTXO as input and creates an UTXO that gives it all to one of the parties.

Assumptions. Timelocks are used to invalidate transactions. That is, a transaction invalidates another one if it spends the same UTXO within its inputs, but has a timelock that is lower by at least Δ . We assume that the original payment channel between Alice and Bob has a timelock of at least $t_0 + \Delta$, and the one between Bob and Charlie has a timelock of at least $t_1 + \Delta$. After tear-down of our construction the timelocks of both channels will be $t_0 - \Delta$ and $t_1 - \Delta$ respectively. We note that this does not make the construction incompatible with pairwise payment channel constructions that do not rely on timelocks for transaction invalidation, such as lightning network style channels. Such channels can perform a state updates using their invalidation method that introduce a timelock before construction, and remove the timelock after tear-down.

Malicious Behavior. Parties abort protocols *Open_VC* and *Close_VC* when they observe another party deviating from the protocol, or if a party delays execution until expiration of the virtual channel, i.e. $t_0 - \Delta$ and $t_1 - \Delta$ respectively.

Open_VC takes an amount of coins $\delta \in \mathbb{N}$ and two pairwise payment channel between three parties as input and creates three new pairwise payment channels, one between each pair of parties. In the following we assume the parties are Alice, Bob and Charlie with payment channels between Alice and Bob, and between Bob and Charlie. Our construction creates a set of transactions as illustrated in Fig. 2. In a nutshell, the purpose of the construction is to allow parties to enforce payout of all of their funds distributed among the offchain channels, while providing fall-back security of their funds in case all other parties misbehave.

First, two Split transactions are created, each spending one of the Funding UTXO that are spent by the original pairwise payment channels. Their timelocks

are t_0 and t_1 respectively s.t. they invalidate the original payment channels. One of the UTXO of each Split transaction contains δ coins and is used as input into a Merge transaction. The other UTXO of each Split transaction is used as Funding UTXO to re-create the original payment channels, albeit each party has $\delta/2$ coins less in these channels. The Merge transaction takes the UTXO with δ coins as input, creates a Funding UTXO for a channel between Alice and Charlie where each possess initially $\delta/2$ coins, and another UTXO gives δ coins to only Bob which represents his collateral. Lastly, two Punish transactions spend the same UTXO as the Merge transaction but give all coins to Alice and Charlie respectively. They have a timelock of $\max(t_0, t_1) + 2\Delta$ such that they are invalidated by the Merge transaction (Fig. 3).

Close_VC takes a virtual channel construction as input and closes them while setting up the original pairwise payment channel but with a balance distribution reflecting the balances in the three payment channel built on-top of the construction. Effectively Alice pays Bob the funds she owes Charlie while Bob forwards these funds to Charlie - and vice versa. The channels have timelocks $t_0 - \Delta$ and $t_1 - \Delta$ respectively to invalidate the Split transactions. Note that a virtual channel construction can only be closed until time $\min(t_0, t_1) - \Delta$ as otherwise the newly constructed payment channels cannot invalidate the Split transactions. Note that having Bob take out $\delta/2$ coins out of both of his original channels within the construction ensures that no party has a negative balance within a pairwise payment channel upon tear-down (Fig. 4).

Enforce_VC lets a party enforce the current state by having it commit a transaction to the blockchain as soon as its timelock expires (Fig. 5).

Atomic Construction. We require that all transactions within our construction are created and respectively invalidated atomically. This is enforced by the order in which transactions are signed. First, parties have to exchange signatures for all transactions except of those spending the original Funding UTXO, i.e. the Split transactions in Open_VC and the root of the pairwise payment channel sub-trees in Close_VC. Afterwards, Alice and Charlie sign these remaining transactions and send the signatures to Bob. Lastly Bob signs them and sends his signatures to Alice and Charlie. Only if a party holds all signatures for all transactions it is involved in, it will consent in performing payments. This ensures security as we will discuss in the following.

Only Alice is Honest. (1) As Bob is the last one to sign, he might interrupt the protocol before Alice receives a signature for the Split transaction. In this case Alice will not consent to any payments and the construction does not change her total balance. Alice can receive her funds by waiting for expiration of her original payment channel's timelock or commitment of the Split transaction by Bob. (2) Bob and Charlie can collude and spend the Funding UTXO that is referenced by their Split transaction. As such the whole transaction sub-tree with the Split transaction as root cannot be committed to the ledger, including the Merge transaction. In that case Alice can commit the Split transaction, and subsequently the Punish transaction. Alice will receive δ coins which is the maximum amount of coins she

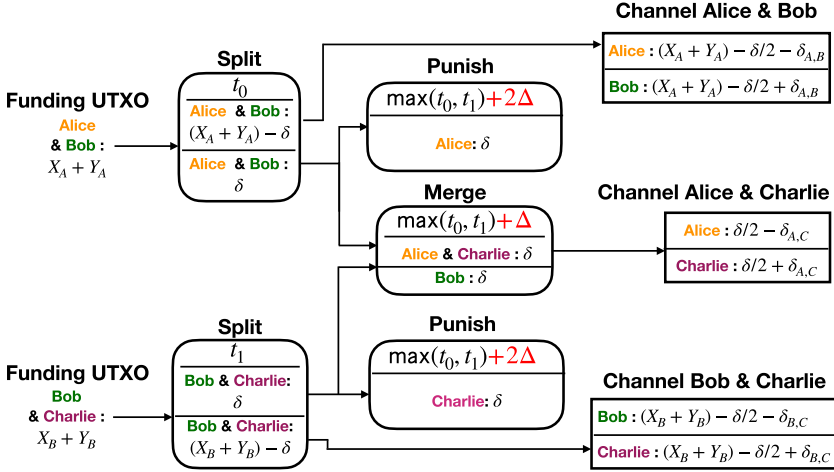


Fig. 2. Overview of the virtual channel construction as a transaction tree. On the left-hand side are Funding UTXO either on the ledger or within previous virtual channels. Boxes with round corners represent the transactions of our construction while the boxes on the right-hand side abbreviate pairwise payment channel’s transaction sub-trees. We omit stating inputs explicitly as they are clear from context.

can receive within her pairwise payment channel with Charlie, as such she does not lose coins. Note that Alice’s channel with Bob is unaffected as it is not within the sub-tree that Bob and Charlie invalidated.

Only Bob is Honest. (1) As Bob is the last to sign transactions, he can assure either both Split transactions are fully signed and they can be committed to the ledger, or none. Moreover he can assure that either both Split transactions will be invalidated upon lockdown or none. (2) Spending the Funding UTXO referenced by the Split transactions always require Bob’s consent by requiring a signature such that Alice and Charlie cannot invalidate any part of the construction’s transaction sub-tree, making Bob to pay out his collateral via a Punish transaction.

Iterative Construction. The pairwise payment channels used as input can either have a Funding UTXO located on a ledger, or a Funding UTXO created by a previous virtual channel construction. In that case timelocks have to be chosen such that within its transaction sub-tree any transaction has a timelock larger than its predecessor’s timelock by at least Δ in order to ensure there is sufficient time to commit them to the ledger. Moreover virtual channel constructions have to be torn-down in reverse order in which they were setup. Iterative constructions requires further analysis of security. The key part to make iterative construction work is the design of the Punish transactions as they secure a party’s funds, including potential collateral payments, while not over-punishing a potentially honest intermediary party: The punishment amount cannot exceed a party’s collateral. Assume the channel between Bob and Charlie is created using a virtual channel construction with channels between Bob and Ingrid and

between Ingrid and Charlie. In that case Ingrid and Charlie can collude by spending their Funding UTXO invalidating the Split transaction between Bob and Charlie making Bob have to pay coins within the Punish transaction between him and Alice. However, these funds as well as the funds Bob has in his channel between him and Charlie are covered by a Punish transaction he has between him and Charlie. Indeed this is the reason why the same amount of coins δ has to be paid into the Merge transaction from both of its Funding UTXO and only those coins are covered by the Punish transaction. This ensures that all funds are covered while not over-punishing the intermediary party in case of iterative virtual channel construction.

Mitigating Wormhole Attacks. Malavolta et al. [13] showed an attack in which two colluding parties skip intermediary parties within a HTLC payment within a payment channel network (1) withholding fees that would have been paid to the intermediary parties and (2) obtaining the fees themselves instead. A variant of this attack could be applied to our construction as we do not require parties to verify that all pairwise payment channel but the ones they participate in were validly constructed. We discuss how to mitigate possible attacks. Although detailed discussion about payout of fees is beyond the scope of this work, we suggest that fees are paid to the intermediary party as compensation for locking up collateral. We note that due to this attacker cannot obtain more fees than they are owed (2). However, attackers could still collude to withhold fees of intermediary parties (1). A mitigation to this attack is that parties would need to proof that such a payment channel was previously constructed, but showing the Funding UTXO that were used and are located on the ledger as well as the whole transaction subtree originating those. A party that receives this information can do a sanity check and store the sub-tree in case they have to do the same proof. This poof serves to show that fees have been paid to the intermediate parties, however, we note that the information might be out-of-date as malicious parties can close their pairwise channel effectively invalidating the whole subtrees.

5 Our Protocols

Here we introduce the constructions for `Open_VC` (Fig. 3), `Close_VC` (Fig. 4) and `Enforce_VC` (Fig. 5) for setup, tear-down and dispute protocols of virtual channels. Due to space constraints these protocols are heavily simplified but derived from the formal protocol $\text{LVPC}_{\text{PWCH}}$ that implements Functionality $\mathcal{F}_{\text{LVPC}, \mathcal{F}_{\text{PWCH}}}$ from Sect. 6. Moreover the complete descriptions can be found in the full version of the work.

In the following protocols we assume that: When executing any protocol all involved honest parties check that execution with the given parameters is permissible, i.e. it will not result in transactions with negative balances, timelocks in the past and that the pairwise payment channel in `Open_VC` or the virtual channel in `Close_VC` are not currently in use with another virtual channel construction. Moreover, for protocols `Open_VC` and `Close_VC` they check that all parties consent execution. Lastly, they abort execution if they observe a party

deviating from the protocol including when their signatures fail verification or when execution times-out. For details we refer to Functionality $\mathcal{F}_{\text{LVPC}, \mathcal{F}_{\text{PWCH}}}$.

Before introducing the protocols, first we define the individual types of transactions used in our construction as well as pairwise payment channel and virtual payment channel.

Algorithm 1 Open Virtual Channel

```

1: function OPEN_VC( $\gamma_0, \gamma_1, \delta$ )
2:    $\text{tr}_{0,S} \leftarrow \text{SPLIT\_TR}(\gamma_0.f, \delta, \gamma_0.t - \Delta)$ 
3:    $\text{tr}_{1,S} \leftarrow \text{SPLIT\_TR}(\gamma_1.f, \delta, \gamma_1.t - \Delta)$ 
4:    $\text{tr}_{0,p} \leftarrow \text{PUNISH\_TR}(\text{OUT\_DELTA}(\text{tr}_{0,S}), \mathcal{P}_A, \max(\gamma_0.t, \gamma_1.t) + \Delta)$ 
5:    $\text{tr}_{1,p} \leftarrow \text{PUNISH\_TR}(\text{OUT\_DELTA}(\text{tr}_{1,S}), \mathcal{P}_C, \max(\gamma_0.t, \gamma_1.t) + \Delta)$ 
6:    $\text{tr}_{\text{mrg}} \leftarrow \text{MERGE\_TR}(\text{OUT\_DELTA}(\text{tr}_{0,S}), \text{OUT\_DELTA}(\text{tr}_{1,S}), \delta, \max(\gamma_0.t, \gamma_1.t))$ 
7:    $(\gamma_{A,B}, \text{tr}_{\text{root},A,B}) \leftarrow \text{open\_virtual}(\text{OUT\_CH}(\text{tr}_{0,S}), \mathcal{P}_A, \mathcal{P}_B,$ 
        $\text{balance}(\gamma_0, \mathcal{P}_A) - \delta/2, \text{balance}(\gamma_0, \mathcal{P}_B) - \delta/2)$ 
8:    $(\gamma_{B,C}, \text{tr}_{\text{root},B,C}) \leftarrow \text{open\_virtual}(\text{OUT\_CH}(\text{tr}_{1,S}), \mathcal{P}_B, \mathcal{P}_C,$ 
        $\text{balance}(\gamma_1, \mathcal{P}_B) - \delta/2, \text{balance}(\gamma_1, \mathcal{P}_C) - \delta/2)$ 
9:    $(\gamma_{A,C}, \text{tr}_{\text{root},A,C}) \leftarrow \text{open\_virtual}(\text{OUT\_CH}(\text{tr}_{\text{mrg}}), \mathcal{P}_A, \mathcal{P}_C, \delta/2, \delta/2)$ 
10:   $\forall$  transactions except Split: Exchange signatures
11:   $\mathcal{P}_A$  &  $\mathcal{P}_C$ : Send Split transactions' signatures to  $\mathcal{P}_B$ 
12:   $\mathcal{P}_B$ : Send Split transactions' signatures to  $\mathcal{P}_A$  &  $\mathcal{P}_C$ 
13:  return  $\gamma^v = (\gamma_0, \gamma_1, \gamma_{A,B}, \gamma_{B,C}, \gamma_{A,C}, \mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C, \delta, \min(\gamma_0.t, \gamma_1.t) - 2\Delta)$ 
14: end function
    
```

Fig. 3. Creation of a virtual channel. Takes two pairwise payment channel γ_0 and γ_1 , and an amount of coins δ as input, and outputs a virtual channel γ^v .

Punish. A Punish transaction takes a Funding UTXO as input but gives all funds to one party. It is parametrized with $(\text{ref}, \mathcal{P}, t_p)$ where ref is a reference to a Funding UTXO f.out , $\mathcal{P} \in \text{f.out.Party}$ is a party and $t_p \in \mathbb{N}$ is a round number.

Algorithm 2 Close Virtual Channel

```

1: function CLOSE_VC( $\gamma^v$ )
2:    $\text{sum}_A = \gamma^v.\gamma_{A,B}.b_A + \gamma^v.\gamma_{A,C}.b_A$ 
3:    $\text{sum}_B = \gamma^v.\gamma_{A,B}.b_B + \gamma^v.\gamma_{A,C}.b_B$ 
4:    $\text{sum}'_B = \gamma^v.\gamma_{B,C}.b_A + \gamma^v.\gamma_{A,C}.b_A$ 
5:    $\text{sum}_C = \gamma^v.\gamma_{B,C}.b_B + \gamma^v.\gamma_{A,C}.b_B$ 
6:    $(\gamma_0, \text{tr}_{\text{root},A,B}) \leftarrow \text{open\_virtual}(\gamma^v.\gamma_0.f, \mathcal{P}_A, \mathcal{P}_B, \text{sum}_A, \text{sum}_B, \gamma^v.t)$ 
7:    $(\gamma_1, \text{tr}_{\text{root},B,C}) \leftarrow \text{open\_virtual}(\gamma^v.\gamma_1.f, \mathcal{P}_B, \mathcal{P}_C, \text{sum}'_B, \text{sum}_C, \gamma^v.t)$ 
8:   $\forall$  transactions except  $\text{tr}_{\text{root},A,B}$  and  $\text{tr}_{\text{root},B,C}$ : Exchange signatures
9:   $\mathcal{P}_A$  signs  $\text{tr}_{\text{root},A,B}$ ,  $\mathcal{P}_C$  signs  $\text{tr}_{\text{root},B,C}$ . Send signatures to  $\mathcal{P}_B$ 
10:  $\mathcal{P}_B$  signs  $\text{tr}_{\text{root},A,B}$  and  $\text{tr}_{\text{root},B,C}$ . Sends signatures to  $\mathcal{P}_A$  and  $\mathcal{P}_C$  respectively
11:  return  $(\gamma_0, \gamma_1)$ 
12: end function
    
```

Fig. 4. Closing of a virtual channel γ^v by recreating the original channels γ_0 and γ_1 . The constructions Split transactions are invalidated by having the roots of the pairwise payment channels have timelocks of at most $\gamma^v.t$.

It is of form $(t_p, \{\text{ref}\}, \{\text{out}\}, \Sigma)$ where $\text{out} = (\text{f_out.b}, \mathcal{P})$. In the following we denote a Punish transaction with these parameter by $\text{PUNISH_TR}(\text{f_out}, \mathcal{P}, t_p)$, and an analogous mapping by $\text{PUNISH_MAP}(\text{f_out}, \mathcal{P})$.

Split. A Split transaction takes a Funding UTXO as input and splits funds across two funding UTXO. It is parametrized with $(\text{ref}, \delta, t_S)$ where ref is a reference to a Funding UTXO f_out , $\delta \in \mathbb{N}, \delta \leq \text{f_out.b}$ is a balance and $t_S \in \mathbb{N}$ is a point in time. It is of form $(t_S, \{\text{ref}\}, \{\text{out}_{ch}, \text{out}_\delta\}, \Sigma)$ where $\text{out}_{ch} = (\text{f_out.b} - \delta, \text{f_out.Party})$ and $\text{out}_\delta = (\delta, \text{f_out.Party})$. In the following we denote a Split transaction with these parameter by $\text{SPLIT_TR}(\text{f_out}, \delta, t_S)$ and an analogous mapping by $\text{SPLIT_MAP}(\text{f_out}, \delta)$. The macros OUT_CH and OUT_DELTA take either a Split transaction or analogous mapping as input and return out_{ch} and out_δ respectively.

Merge. A Merge transaction takes two funding UTXO by three parties and creates a new Funding UTXO. It is parametrized with $(t_M, \text{f_out}_A, \text{f_out}_B, b)$ where $t_M \in \mathbb{N}$ is a round number, $\text{f_out}_A, \text{f_out}_B$ are two Funding UTXO and $b \in \mathbb{N}, b = \text{f_out}_A.b = \text{f_out}_B.b$ is an amount of coins. Moreover for the involved parties $\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C$ holds $\mathcal{P}_A, \mathcal{P}_B \in \text{f_out}_A.\text{Party}, \mathcal{P}_B, \mathcal{P}_C \in \text{f_out}_B.\text{Party}$. Given, $\text{out}_{ch} = (b, \{\mathcal{P}_A, \mathcal{P}_C\})$ and $\text{out}_B = (b, \{\mathcal{P}_B\})$, then a Merge transaction is of the form $(t_M, \{\text{ref}(\text{f_out}_A), \text{ref}(\text{f_out}_B)\}, \{\text{out}_{ch}, \text{out}_B\}, \Sigma)$. We denote a Merge transaction with these parameter by $\text{MERGE_TR}(\text{f_out}_A, \text{f_out}_B, b, t_M)$ and an analogous mapping by $\text{MERGE_MAP}(\text{f_out}_A, \text{f_out}_B, b)$. The macro OUT_CH takes a Merge transaction or analogous mapping as input and returns out_{ch} .

Algorithm 3 Enforce Virtual Channel

```

1: function ENFORCE_VC( $\gamma^v$ )
2:   for all tr in transactions of  $\gamma^v$  do
3:     if tr.t <  $\tau$  &  $\forall o \in \text{tr.In} : o$  is on the ledger then
4:       Commit tr to the ledger
5:     end if
6:   end for
7: end function

```

Fig. 5. Parties enforce the state presented by the virtual payment channel construction by committing transactions to the ledger whenever possible, i.e. as soon as their timelocks expire and UTXO referenced in their inputs are present on the ledger.

Function $\text{open_virtual}(f, \mathcal{P}, \mathcal{P}', b, b', t)$ is used to open a pairwise payment channel with the provided, Funding UTXO f , between the two parties $\mathcal{P}, \mathcal{P}'$, respective balance distribution b, b' and optional timelock t . See the full version of this paper for details.

Definition 1. A pairwise payment channel γ is a tuple of form $\gamma = (\text{id}, f, \mathcal{P}_A, \mathcal{P}_B, b_A, b_B, t, t_0)$ where $\text{id} \in \mathbb{N}$ is a unique identifier, f is a funding UTXO, $\mathcal{P}_A, \mathcal{P}_B$ are parties, $b_A, b_B \in \mathbb{N}$ are balances of $\mathcal{P}_A, \mathcal{P}_B$ respectively.

Definition 2. A lightweight virtual payment channel γ^v is a tuple of form $(\text{id}, \gamma_0, \gamma_1, \gamma_{A,B}, \gamma_{B,C}, \gamma_{A,C}, \mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C, \delta, t)$ where $\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C$ are three parties, γ_0, γ_1 are pairwise payment channel between $\mathcal{P}_A, \mathcal{P}_B$ and $\mathcal{P}_B, \mathcal{P}_C$ respectively provided as inputs, $\gamma_{A,B}, \gamma_{B,C}, \gamma_{A,C}$ are pairwise payment channel created by the construction between each pair of parties, δ is the capacity of channel $\gamma_{A,C}$ between $\mathcal{P}_A, \mathcal{P}_C$ and $t \in \mathbb{N}$ is a point in time until which the channel can be closed.

For simplicity we omit stating id explicitly when using pairwise or lightweight virtual payment channels.

6 The Ideal Functionality

In the following we present formal treatment of our protocol in the UC framework by introducing a functionality for lightweight virtual payment channel $\mathcal{F}_{\text{LVPC}, \mathcal{F}_{\text{PWCH}}}$, associated with functionality $\mathcal{F}_{\text{PWCH}}$. For this we make use of auxiliary functionality $\mathcal{F}_{\text{Script}}$, global UTXO ledger functionality $\mathcal{G}_{\text{UTXO-Ledger}}$, global clock functionality $\mathcal{G}_{\text{Clock}}$, and functionality \mathcal{F}_{SIG} . These functionalities are defined in Appendix A.

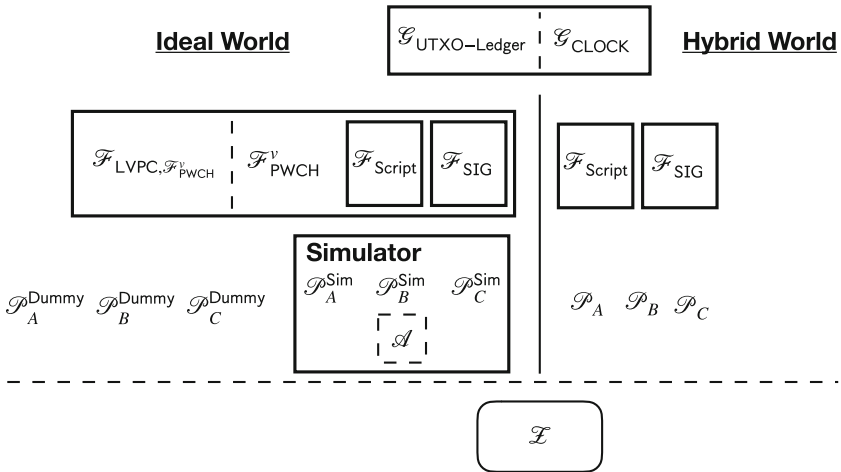


Fig. 6. Overview of our setup within the UC framework.

Overview. Figure 6 depicts an overview of our construction. The setting is split up in a Ideal world and a $(\mathcal{G}_{\text{CLOCK}}, \mathcal{G}_{\text{UTXO-Ledger}}, \mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{Script}})$ - Hybrid World. The global functionality $\mathcal{G}_{\text{UTXO-Ledger}}$ is associated with the global $\mathcal{G}_{\text{CLOCK}}$ functionality and accessible from either world. The lightweight virtual channel functionality $\mathcal{F}_{\text{LVPC}}$ is associated with the pairwise payment channel functionality $\mathcal{F}_{\text{PWCH}}$ receiving access to its internal state and helper functions. $\mathcal{F}_{\text{PWCH}}$ includes and replicates the interfaces and behavior of $\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{Script}}$.

The Ideal Virtual Channel Functionality. The lightweight virtual payment channel functionality $\mathcal{F}_{\text{LVPC}, \mathcal{F}_{\text{PWCH}}}$ is used to create and close virtual payment channel between three parties. It provides access to functions VC-OPEN, VC-Close and VC-Enforce. Function VC-OPEN takes two pairwise payment channel between three parties as input, disables state updates on those, and creates three new pairwise payment channel, one between each pair of parties. To be able to enforce these channels it creates and stores mappings that represent Split, Merge and Punish transactions together with the time at which they become valid. Function VC-Close takes a virtual channel as input. First it checks whether no virtual channel have been created using the pairwise payment channel created with it. If positive it disables state updates on these channels, re-enables state updates for the original channels, updates their balance to reflect the latest balance distribution among the three channels and sets the channel's timelocks to be lower than the one of the Split mappings by Δ . Function Enforce is used to commit any mapping representing Split, Merge or Punish transactions if their timelocks have expired. This disables closure of the virtual channel because the funding UTXO of the original pairwise payment channels are removed from the ledger.

General Behavior. Below we describe $\mathcal{F}_{\text{LVPC}, \mathcal{F}_{\text{PWCH}}}$ non function-specific behavior first before detailing its interface.

Update time: At beginning of each round in which functionality is activated send message $(\text{clock-read}, \text{sid})$ to $\mathcal{G}_{\text{CLOCK}}$ and receive the reply $(\text{CLOCK-READ}, \text{sid}, \tau')$. Set internal variable $\tau = \tau'$.

Interactions with simulator: Whenever the functionality receives a message msg from any party or from $\mathcal{G}_{\text{UTXO-Ledger}}$ it leaks the message to the simulator and appends sender and receiver.

Synchronization with the simulation: Interactions with the ledger are used to read its state as well as trigger a state change. The state on the ledger as well as whether a state change is permissible depends on the moment they are done as

transactions that change the set of UTXO on the ledger can be sent by a party at any time. Therefore we need to ensure that the functionality's interaction with the ledger are at the same time as they happen in the simulation to achieve the same results and receive the same replies. Whenever the functionality sends a message msg to the ledger it waits for the simulator to leak a similar message by a honest party. Note that a TRANSACTION tagged message from the simulator is processed by the $\mathcal{F}_{\text{Script}}$ functionality first. Then the functionality sends the message only once and forwards any replies to the simulator.

Handling corrupted parties: We assume static corruption by a malicious adversary. At the beginning of execution the functionality asks the simulator which parties are controlled by the adversary and stores this information in set COR. The functionality ignores requests from any party in the ideal world of which counterpart in the simulation is corrupted by the adversary. The functionality needs to learn whether a party corrupted by the adversary misbehaved or delayed execution of a protocol beyond a channel's lifetime. For this matter as soon as the simulator leaks that any simulated honest party \mathcal{P}'_h sends message (FAILURE, sid , msg) to \mathcal{Z} the functionality aborts execution of the function triggered by receiving msg and sends (FAILURE, sid , msg) to \mathcal{P}'_h 's dummy-party counterpart \mathcal{P}_h in the ideal world.

Functionality $\mathcal{F}_{\text{LVPC}, \mathcal{F}_{\text{PWCH}}}$

Has access to $\mathcal{F}_{\text{PWCH}}$'s helper functions.

State: Set of closable virtual payment channel Γ^v of virtual payment channels. List M^v of entries (γ^v, m, t) where γ^v is a virtual payment channel, m is a partial mapping and t is a round number. It has access to the internal state of $\mathcal{F}_{\text{PWCH}}^v$ including set Γ of pairwise payment channels. Moreover it shares common state with $\mathcal{F}_{\text{PWCH}}^v$ which is the current round number τ , list of corrupted parties COR and set of consent giving parties CONS.

Initialization: Initializes $\mathcal{F}_{\text{PWCH}}^v$ and shared state τ , COR, CONS. Sets $\Gamma^v = M^v = \emptyset$.

VC-Open: Execute upon receiving message $\text{msg} = (\text{OPEN}, \text{sid}, \gamma_0, \gamma_1, \delta, t, \mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C)$ from $\mathcal{P} \in \{\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C\}$ where $\gamma_0, \gamma_1 \in \Gamma$, $\delta \in \mathbb{N}$ is an amount of coins and $t \in \mathbb{N}$ is a round number. Let $\text{Parties}_h = \{\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C\} \setminus \text{COR}$:

1. **if** $\text{cns} = \text{consent}(\mathcal{P}, \{\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C\}, \text{msg}) = \text{no_consent}$: halt
 2. Verify; if any verification fails send $(\text{FAILURE}, \text{sid}, \text{msg})$ to all in \mathcal{P}_h :
 - $\{\mathcal{P}_A, \mathcal{P}_B\} = \{\gamma_0.\mathcal{P}_A, \gamma_0.\mathcal{P}_B\}$ and $\{\mathcal{P}_B, \mathcal{P}_C\} = \{\gamma_1.\mathcal{P}_A, \gamma_1.\mathcal{P}_B\}$
 - **for each** $\gamma \in \{\gamma_0, \gamma_1\}$: **if** $\{\gamma.\mathcal{P}_A, \gamma.\mathcal{P}_B\} \setminus \text{COR} \neq \emptyset$ **then**
 - $\gamma \in \Gamma$; $\gamma.t > \tau + 2\Delta$; $\gamma.t_0 > \tau + 2\Delta$; $\gamma.b_A \geq \delta/2$ and $\gamma.b_B \geq \delta/2$
- Upon receiving message $(\text{SUCCESS}, \text{sid}, \text{msg})$ from all parties in Parties_h :

1. Create Mappings
 - Split: $m_{0,S} = \text{SPLIT_MAP}(\gamma_0.f, \delta)$; $m_{1,S} = \text{SPLIT_MAP}(\gamma_1.f, \delta)$
 - Merge: $m_{\text{mrg}} = \text{MERGE_MAP}(\text{OUT_DELTA}(m_{0,S}), \text{OUT_DELTA}(m_{1,S}), \delta)$
 - Punish: $m_{0,p} = \text{PUNISH_MAP}(\text{OUT_DELTA}(m_{0,S}), \mathcal{P}_A)$
 - $m_{1,p} = \text{PUNISH_MAP}(\text{OUT_DELTA}(m_{1,S}), \mathcal{P}_C)$
2. Create new channel, revoke old
 - $\gamma_{A,B} = \text{open_virtual}(f_{A,B}, \mathcal{P}_A, \mathcal{P}_B, \text{balance}(\gamma_0, \mathcal{P}_A) - \delta/2, \text{balance}(\gamma_0, \mathcal{P}_B) - \delta/2, t, \gamma_0.t)$
 - $\gamma_{B,C} = \text{open_virtual}(f_{B,C}, \mathcal{P}_B, \mathcal{P}_C, \text{balance}(\gamma_1, \mathcal{P}_B) - \delta/2, \text{balance}(\gamma_1, \mathcal{P}_C) - \delta/2, t, \gamma_1.t)$
 - $\gamma_{A,C} = \text{open_virtual}(f_{A,C}, \mathcal{P}_A, \mathcal{P}_C, \delta/2, \delta/2, t, \max(\gamma_0.t, \gamma_1.t) + \Delta)$
 - Revoke old channel: $\text{revoke}(\gamma_0)$ and $\text{revoke}(\gamma_1)$
3. Virtual channel: $\gamma^v = (\gamma_0, \gamma_1, \gamma_{A,B}, \gamma_{B,C}, \gamma_{A,C}, \mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C, \delta, \min(\gamma_0.t, \gamma_1.t) - 2\Delta)$
4. Update: $\Gamma^v = \Gamma \cup \{\gamma^v\}$ and $M^v = M \cup \{(\gamma^v, m_{0,S}, \gamma_0.t - \Delta), (\gamma^v, m_{1,S}, \gamma_1.t - \Delta), (\gamma^v, m_{\text{mrg}}, \max(\gamma_0.t, \gamma_1.t)), (\gamma^v, m_{0,p}, \max(\gamma_0.t, \gamma_1.t) + \Delta), (\gamma^v, m_{1,p}, \max(\gamma_0.t, \gamma_1.t) + \Delta)\}$
5. Return message $(\text{SUCCESS}, \text{sid}, \text{msg})$ to \mathcal{P}

VC-Close: Upon receiving message $\text{msg} = (\text{CLOSE}, \text{sid}, \gamma^v,)$ from $\mathcal{P} \in \{\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C\}$ where $\mathcal{P}_A = \gamma^v.\mathcal{P}_A$, $\mathcal{P}_B = \gamma^v.\mathcal{P}_B$ and $\mathcal{P}_C = \gamma^v.\mathcal{P}_C$. Let $(\gamma_0, \gamma_1, \gamma_{A,B}, \gamma_{B,C}, \gamma_{A,C}, \mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C, \delta, t) = \gamma^v$. Moreover let $\text{Parties}_h = \{\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C\} \setminus \text{COR}$. Do:

1. **if** $\text{cns} = \text{consent}(\mathcal{P}, \{\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C\}, \text{msg}) = \text{no_consent}$: halt
2. Verify; if any verification fails send $(\text{FAILURE}, \text{sid}, \text{msg})$ to all in \mathcal{P}_h :
 - $\gamma^v \in \Gamma^v$; $\{\gamma_{A,B}, \gamma_{B,C}, \gamma_{A,C}\} \subseteq \mathcal{F}_{\text{PWCH}}^v \Gamma$; $t > \tau$; $\gamma_0.t_0 > \tau + 2\Delta$, $\gamma_1.t_0 > \tau + 2\Delta$

Upon receiving message $(\text{SUCCESS}, \text{sid}, \text{msg})$ from all parties in Parties_h :

1. Revoke old channel, reactivate and update original channel:
 - $\text{activate}(\gamma_0)$, $\text{activate}(\gamma_1)$
 - $\text{state.update}(\gamma_0, \mathcal{P}_A, b_A, \mathcal{P}_B, b_B, 2\Delta)$; $\text{state.update}(\gamma_1, \mathcal{P}_B, b'_B, \mathcal{P}_C, b_C, 2\Delta)$
 where $b_A = \gamma_{A,B}.b_A + \gamma_{A,C}.b_A$; $b_B = \gamma_{A,B}.b_B + \gamma_{A,C}.b_B$; $b'_B = \gamma_{B,C}.b_A + \gamma_{A,C}.b_A$; $b_C = \gamma_{B,C}.b_B + \gamma_{A,C}.b_B$.
 - $\text{revoke}(\gamma_{A,B})$, $\text{revoke}(\gamma_{B,C})$, $\text{revoke}(\gamma_{A,C})$

2. Update internal state: $\Gamma^v = \Gamma^v \setminus \{\gamma^v\}$
 3. Return message (SUCCESS, sid, msg) to \mathcal{P}
- VC-Enforce:** Triggered upon receiving $msg = (\text{ENFORCE}, sid, \gamma^v)$ from party \mathcal{P} where γ^v is a lightweight virtual payment channel. Let $\mathcal{P}_A = \gamma^v.\mathcal{P}_A$, $\mathcal{P}_B = \gamma^v.\mathcal{P}_B$ and $\mathcal{P}_C = \gamma^v.\mathcal{P}_C$.
1. Check $\mathcal{P} \in \{\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C\}$, $\gamma \in \Gamma$
 2. Let $M_\gamma = \{(\gamma^v, m', t') \mid (\gamma^v, m', t') \in M^v; \gamma^v = \gamma^v; t' \leq \tau, \exists \text{utxo} \in m'.\text{In} : \mathcal{P} \in \text{utxo.Party}\}$
 3. **for each** $m \in M_\gamma$
 - Let (In, Out) = m . For all $o \in \text{In}$ send message (CHECK, sid, o) to $\mathcal{G}_{\text{UTXO-Ledger}}$. If it replies (CHECK_OKAY, sid, o) for all $o \in \text{In}$:
 - Send message (TRANSACTION, sid, m) to $\mathcal{G}_{\text{UTXO-Ledger}}$
 - Channel cannot be closed: $\Gamma^v = \Gamma^v \setminus \{\gamma^v\}$
 4. Return message (SUCCESS, sid, msg) to \mathcal{P}

Definition 3. *Balance Security: The sum of a honest party's funds only changes with its consent.*

Definition 4. *Liveness: Eventually all of a party's funds are unlocked and committed to the ledger within UTXO that are spendable by the party alone.*

Security of Funds and Liveness. In the following we briefly argue that functionality $\mathcal{F}_{\text{LVPC}, \mathcal{F}_{\text{PWCH}}}$ fulfills these two properties for honest parties by design. We expect a honest party to call sub-function VC-Enforce as soon as they would lead to submission of a mapping to the ledger, i.e. at times $\gamma^v.\gamma_0.t - \Delta$, $\gamma^v.\gamma_1.t - \Delta$, $\max(\gamma^v.\gamma_0.t, \gamma^v.\gamma_1.t)$ and in case a punish transaction has to be committed at time $\max(\gamma^v.\gamma_0.t, \gamma^v.\gamma_1.t) + \Delta$. Eventually all funds that a honest party holds will be accessible over UTXOs on the ledger such that liveness holds. Balance Security holds since only $\mathcal{F}_{\text{PWCH}}$'s channel update function, which requires the party's consent, changes a honest parties' balance.

Theorem 1. *Protocol LVPC_{PWCH} realizes $\mathcal{F}_{\text{LVPC}, \mathcal{F}_{\text{PWCH}}}$ in the $(\mathcal{G}_{\text{CLOCK}}, \mathcal{G}_{\text{UTXO-Ledger}}, \mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{Script}})$ - Hybrid World.*

Proof: Due to space constraints we refer to the paper's full version.

7 Future Work

We use timelocks to create an order in which transactions in our constructions are valid. However, different techniques for invalidating transactions or replacing transactions offchain might be used instead to have less restrictions on the lifetime of a virtual channel. For this we can adapt techniques as introduced for the Lightning Network [15] or ulti [14].

Lastly we argue that our construction provides incentive for research into route discovery protocols that yield multiple paths.

A Additional Functionalities and Protocols

Functionality $\mathcal{G}_{\text{CLOCK}}$ [10,11,12]

The functionality is accessible by any entity and associated with a global functionality $\mathcal{G}_{\text{UTXO-Ledger}}$.

State: Stores time $\tau \in \mathbb{N}$, a set of parties \mathcal{P} , bit $d_{\mathcal{G}_{\text{UTXO-Ledger}}} \in \{0, 1\}$ as well as bits $d_{\mathcal{P}}$ for each party in \mathcal{P} .

Initialization: Sets $\tau = d_{\mathcal{G}_{\text{UTXO-Ledger}}} = 0$ and $\mathcal{P} = \emptyset$.

Register: Upon receiving message $(\text{register}, \text{sid})$ from party \mathcal{P} , set $\mathcal{P} = \mathcal{P} \cup \{\mathcal{P}\}$, store a bit $d_{\mathcal{P}} \in \{0, 1\}$ initialized with $d_{\mathcal{P}} = 0$ and send message $(\text{REGISTER}, \text{sid}, \mathcal{P})$ to the adversary.

Clock Update Ledger: Upon receiving message $(\text{clock-update}, \text{sid})$ from $\mathcal{G}_{\text{UTXO-Ledger}}$ set $d_{\mathcal{G}_{\text{UTXO-Ledger}}} = 1$ and send message $(\text{CLOCK-UPDATE}, \text{sid}, \mathcal{P})$ to the adversary.

Clock Update Party: Upon receiving message $(\text{clock-update}, \text{sid})$ from party \mathcal{P} set $d_{\mathcal{P}} = 1$. If $d_{\mathcal{G}_{\text{UTXO-Ledger}}} = 1$ and $d_{\mathcal{P}} = 1$ for all honest parties in \mathcal{P} , set $\tau = \tau + 1$, $d_{\mathcal{G}_{\text{UTXO-Ledger}}} = 0$ and $d_{\mathcal{P}} = 0$ for all honest parties in \mathcal{P} . Lastly send message $(\text{CLOCK-UPDATE}, \text{sid}, \mathcal{G}_{\text{UTXO-Ledger}})$ to the adversary.

Clock Read: Upon receiving message $(\text{clock-read}, \text{sid})$ from any entity reply with message $(\text{CLOCK-READ}, \text{sid}, \tau)$.

The Global Functionality $\mathcal{G}_{\text{UTXO-Ledger}}$ models a UTXO based ledger maintaining a publicly readable set of UTXO. The differences between $\mathcal{G}_{\text{UTXO-Ledger}}$ and the ledger functionality by Kiayias et al. [12] are twofold. For one instead of using a verification predicate to check the validity of transactions, we move this verification into a second functionality $\mathcal{F}_{\text{Script}}$ representing required parts of a blockchains scripting language similar to the separation of ledger and smart contract functionalities in the work of Dziembowski et al. [8,9]. For another we explicitly make use of UTXO which is required for our construction.

Functionality $\mathcal{G}_{\text{UTXO-Ledger}}$

State: Stores set of UTXO \mathcal{U} .

Initialization: \mathcal{Z} sends the initial state \mathcal{U}_0 . Sets $\mathcal{U} := \mathcal{U}_0$.

Additional interface: The functionality wraps the $\mathcal{F}_{\text{Script}}$ and \mathcal{F}_{SIG} functionalities internally and replicates their interface. Any messages to these functionalities are processed according to their definition.

Transaction: Upon receiving, from either \mathcal{Z} or functionalities, the message $(\text{TRANSACTION}, \text{sid}, M)$ where M is a partial UTXO mapping, do : Let $(\text{In}, \text{Out}) = M$. Check that $\text{In} \subseteq \mathcal{U}$, $\sum_{i \in \text{In}} i.b \geq \sum_{o \in \text{Out}} o.b$. Upon success within Δ rounds set $\mathcal{U} = (\mathcal{U} \setminus \text{In}) \cup \text{Out}$.

Check UTXO: Upon receiving $(\text{CHECK}, \text{sid}, \text{out})$ where $\text{out} \in \text{Output}$ reply $(\text{CHECK_OKAY}, \text{sid}, \text{out})$ if $\text{out} \in \mathcal{U}$ and $(\text{CHECK_FAILURE}, \text{sid}, \text{out})$ otherwise.

Functionality \mathcal{F}_{SIG} [2]

State: Stores set K which contain tuples of form (\mathcal{P}, v) where \mathcal{P} is a party and v is a verification key. Set S with entries of form (m, σ, v, b) where m is a message, σ as signature, v a verification key and $b \in \{0, 1\}$.

Key Generation: Upon receiving message (KeyGen, sid) from party \mathcal{P} verify $sid = (\mathcal{P}, sid')$ for some sid' . If that is the case hand (KeyGen, sid) to adversary. Upon receiving $(\text{VERIFICATIONKEY}, sid, v)$ from the adversary, forward the message to \mathcal{P} and store (\mathcal{P}, v) in K .

Signature Generation: Upon receiving message (SIGN, sid, m) from a party \mathcal{P} verify that $sid = (\mathcal{P}, sid')$ for some sid' . If that is the case, send (SIGN, sid, m) to adversary. Upon receiving $(\text{SIGNATURE}, sid, m, \sigma)$ from the adversary, if $(m, \sigma, v, 0) \notin S$ send an error message to \mathcal{P} and halt. Otherwise store $(m, \sigma, v, 0)$ in S and send $(\text{SIGNATURE}, sid, m, \sigma)$ to \mathcal{P} .

Signature Verification: Upon receiving message $(\text{VERIFY}, sid, m, \sigma, v')$ from a party \mathcal{P} forward it to the adversary. Upon receiving $(\text{VERIFIED}, sid, m, \phi)$ from the adversary do:

1. If $v' = v$ and $(m, \sigma, v, 1) \in S$ set $f = 1$.
2. Else if $v' = v$, $(m, \sigma', v, 1) \notin S$ for any σ' and \mathcal{P} is not corrupted by the adversary, store $(m, \sigma, v, 0)$ in S and set $f = 0$.
3. Else if $(m, \sigma, v', f') \in S$ for any v', f' set $f = f'$.
4. Else store (m, σ, v', ϕ) in S and set $f = \phi$.

Send $(\text{VERIFIED}, sid, m, f)$ to \mathcal{P} .

Functionality $\mathcal{F}_{\text{Script}}$ (Multi-Signatures, Timelocks)

State: Stores set K with entries of form (\mathcal{P}, v) where \mathcal{P} is a party and v is a verification key.

Registering Verification Key: Upon receiving $(\text{VERIFICATIONKEY}, sid, v)$ from a party \mathcal{P} store (\mathcal{P}, v) in K .

Transaction: Upon receiving $(\text{TRANSACTION}, sid, tr)$ from \mathcal{P} , let $(\text{In}, \text{Out}, t, \Sigma) = tr$ and $\text{stub} = \text{In}, \text{Out}, t$.

- Update time: Send $(\text{GET-TIME}, sid, \cdot)$ to $\mathcal{G}_{\text{Clock}}$ and receive $(\text{GET-TIME}, sid, \tau)$
- Verify that $\forall \text{utxo} \in \text{In}: t \leq \tau$. Halt, otherwise.
- Verify $\forall \text{utxo} \in \text{In}$: For each $\mathcal{P} \in \text{utxo.Party}$ retrieve (\mathcal{P}, v) from K . Verify that Σ contains a signature of stub from \mathcal{P} . For each $\sigma \in \Sigma$ send $(\text{VERIFY}, sid, \text{stub}, \sigma, v)$ to \mathcal{F}_{Sig} and verify that \mathcal{F}_{SIG} replies with $(\text{VERIFIED}, sid, \text{stub}, 1)$ exactly once.
- Send $(\text{TRANSACTION}, sid, \text{Removes}, \text{Adds})$ to $\mathcal{G}_{\text{UTXO-Ledger}}$

References

1. 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>
2. Canetti, R.: Universally composable signature, certification, and authentication. In: Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004, pp. 219–233. IEEE (2004)
3. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 61–85. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_4
4. Chakravarty, M.M., Chapman, J., MacKenzie, K., Melkonian, O., Jones, M.P., Wadler, P.: The extended UTXO model. In: 4th Workshop on Trusted Smart Contracts (2020)
5. Chakravarty, M.M., et al.: Hydra: Fast isomorphic state channels. Cryptology ePrint Archive, Report 2020/299. <https://eprint.iacr.org/2020/299>
6. Croman, K., et al.: On scaling decentralized blockchains. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D., Brenner, M., Rohloff, K. (eds.) FC 2016. LNCS, vol. 9604, pp. 106–125. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_8
7. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Pelc, A., Schwarzmann, A.A. (eds.) SSS 2015. LNCS, vol. 9212, pp. 3–18. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21741-3_1
8. Dziembowski, S., Ekey, L., Faust, S., Malinowski, D.: PERUN: Virtual payment channels over cryptographic currencies. Cryptology ePrint Archive, Report 2017/635 (2017). <http://eprint.iacr.org/2017/635>
9. Dziembowski, S., Faust, S., Hostáková, K.: General state channel networks. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018, pp. 949–966. ACM Press (2018). <https://doi.org/10.1145/3243734.3243856>
10. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 477–498. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_27
11. Kiayias, A., Litos, O.S.T.: A composable security treatment of the lightning network. IACR Cryptol. ePrint Archive **2019**, 778 (2019)
12. Kiayias, A., Zhou, H.S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology - EUROCRYPT 2016, pp. 705–734. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_25
13. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: NDSS (2019)
14. PDecker, C., Russel, R., Osuntokun, O.: eltoo: A simple layer2 protocol for bitcoin. See <https://blockstream.com/eltoo.pdf> (2017)
15. Poon, J., Dryja, T.: The bitcoin lightning network: scalable off-chain instant payments. See <https://lightning.network/lightning-network-paper.pdf> (2016)

Privacy-Enhancing Tools



Chosen-Ciphertext Secure Multi-identity and Multi-attribute Pure FHE

Tapas Pal^(✉) and Ratna Dutta

Department of Mathematics, Indian Institute of Technology Kharagpur,
Kharagpur, India
tapas.pal@iitkgp.ac.in, ratna@maths.iitkgp.ernet.in

Abstract. A *multi-identity pure fully homomorphic encryption* (MIFHE) enables a server to perform arbitrary computation on the ciphertexts that are encrypted under different identities. In case of *multi-attribute pure FHE* (MAFHE), the ciphertexts are associated with different attributes. Clear and McGoldrick (CANS 2014) gave the first chosen-plaintext attack secure MIFHE and MAFHE based on indistinguishability obfuscation. In this study, we focus on building MIFHE and MAFHE which are secure under type 1 of chosen-ciphertext attack (CCA1) security model. In particular, using witness pseudorandom functions (Zhandry, TCC 2016) and multi-key pure FHE or MFHE (Mukherjee and Wichs, EUROCRYPT 2016) we propose the following constructions:

- CCA secure *identity-based encryption* (IBE) that enjoys an *optimal* size ciphertexts, which we extend to a CCA1 secure MIFHE scheme.
- CCA secure *attribute-based encryption* (ABE) having an optimal size ciphertexts, which we transform into a CCA1 secure MAFHE scheme.

By optimal size, we mean that the bit-length of a ciphertext is the bit-length of the message plus a security parameter multiplied with a constant. Known constructions of multi-identity(attribute) FHEs are either leveled, that is, support only bounded depth circuit evaluations or secure in a weaker CPA security model. With our new approach, we achieve both CCA1 security and evaluation on arbitrary depth circuits for multi-identity(attribute) FHE schemes.

Keywords: Witness pseudorandom function · Identity-based encryption · Attribute-based encryption · Fully homomorphic encryption

1 Introduction

Gentry settled the open problem of computing on encrypted data by proposing the first fully homomorphic encryption (FHE) [17] scheme based on ideal lattices. Afterwards, many researchers developed improved variants of Gentry's FHE [6, 7, 31]. These are all *leveled* FHE where a bounded depth circuit can be evaluated on encrypted data. While the error in an evaluated ciphertext may blow up with increasing depth, Gentry's bootstrapping technique [17] can be applied to

convert any leveled FHE into a *pure* FHE which handles arbitrary depth circuits. The bootstrapping relies on circular security means that the scheme is secure even when the adversary is given an encryption of the secret-key.

Identity-based encryption (IBE) [3] gives us the freedom to encrypt data using any arbitrary string (treated as identity) instead of a specified public-key. Constructing identity-based FHE (IBFHE) remained difficult due to the presence of evaluation key until Gentry, Sahai and Waters [18] built a leveled FHE based on learning with errors (LWE) where the public parameters serve the role of the evaluation key. Compiling existing LWE-based identity-based encryption or LWE-based attribute-based encryption (ABE) with their FHE, [18] came up with efficient IBFHE and attribute-based FHE (ABFHE). Clear et al. [12] extends the IBFHE of [18] to multi-identity setting where evaluation can be performed with multiple users data and decryption requires a collaboration of their secret-keys. However, Gentry's bootstrapping theorem can not be applied to convert a leveled IBFHE (or ABFHE) into a pure IBFHE (or pure ABFHE). Since evaluation requires encryption of the secret-keys under the respective identities, the transformed IBFHE becomes interactive which is noted as *weak* [7].

To build a pure IBFHE, Clear and McGoldrick [11] used indistinguishability obfuscation ($i\mathcal{O}$) [30] and a pure FHE scheme. Specifically, they utilized the punctured technique of [30] to create a unique public-secret FHE key pair corresponding to an identity. The IBFHE can be extended to multi-identity pure FHE (MIFHE) when we use a multi-key pure FHE (MFHE) [27] in place of the normal FHE. The work [11] also described a multi-attribute pure FHE (MAFHE) using $i\mathcal{O}$. MAFHE enables us to encrypt messages under different attributes instead of users identities. A generic construction of (almost pure) MAFHE with a bounded number of parties was given in [10] which employs a MFHE and a leveled multi-attribute FHE.

All existing constructions of MIFHE or MAFHE [8,11] are either CPA secure or based on a powerful primitive $i\mathcal{O}$. In case of leveled variants of those primitives [5,12,18], known constructions have started from LWE-based IBE or ABE which mostly provide security in CPA model, hence the corresponding FHEs are inherently CPA secure. It is trivial to observe that CCA security can not be realized for FHE like primitives as evaluation is a public algorithm. But, we can still consider CCA1 security where the adversary is given access to the decryption oracle up-until it receives the challenge ciphertext. Canetti et al. [8] gave a generic construction of CCA1 secure MFHE from a CPA secure MIFHE and instantiated their (pure) MIFHE based on sub-exponential $i\mathcal{O}$. So we ask: *Can we build CCA1 secure MIFHE or MAFHE? Can we construct these primitives without using obfuscation?*

In this paper, we find out affirmative answers to those questions. Recently, Zhandry introduced a different type of pseudorandom function (PRF), called witness PRF (WPRF) [33], which can produce a pseudorandom value $y = F(\text{fk}, x)$ corresponding to an NP statement x using a secret function key fk and anyone holding a valid witness of x can recompute y using a public evaluation key ek . If a statement x does not belong to the NP language then y becomes indis-

tinguishable from random. The primitive finds many applications in building cryptographic tools such as non-interactive multiparty key exchange, witness encryption (WE), poly-many hardcore bits for one-way functions (OWFs) [33] that are previously possible only from $i\mathcal{O}$. We aim to construct CCA1 secure MIFHE and MAFHE schemes using WPRF.

Zhandry [33] built WPRF from multilinear subset-sum Diffie-Hellman assumption which is a *target-group* assumption and hence most of the existing (source-group based) attacks on multilinear maps may not be a threat to the WPRF. On the other side, WPRF construction of [28] based on sublinear compact randomized encoding and puncturable PRF indicates that it belongs to obfustopia. However, WPRF is not known to imply $i\mathcal{O}$ and seems to be a much weaker assumption than $i\mathcal{O}$ [33]. Few primitives like *smooth projective hash functions* [13], *functional PRFs* [4] and *publicly evaluable PRFs* [9] that are close to the notion of WPRF have already been realized from standard assumptions. Therefore, it is more likely to realize WPRF from standard assumptions much before the community arrive at a practical construction of $i\mathcal{O}$.

Contribution. This work investigates applications of WPRF in identity-based and attribute-based cryptography.

1. Multi-identity Pure FHE: In the era of cloud computing, it is highly desirable to run arbitrarily complex programs over any type of encrypted data. To compute on the ciphertexts of an IBE scheme, we build the first CCA1 secure MIFHE using WPRF and MFHE. The stepping-stone of our MIFHE is a CCA secure IBE that we construct from WPRF and a special signature scheme.

Our goal is to use OWFs along with WPRF to get a CCA secure IBE with short secret-keys and optimal size ciphertexts. In particular, we take a pseudorandom generator (PRG) and a secure signature scheme both of which can be efficiently realized from OWFs [29]. First we generate a pair of WPRF keys (fk, ek) for an NP language $L = \{(\text{id}, v, \text{vk}) : (\exists u \text{ such that } \text{PRG}(\text{id} \oplus u) = v) \text{ or } (\exists \sigma \text{ such that } \text{Vrfy}(\text{vk}, \text{id}, \sigma) = 1)\}$ with a relation R where id is an identity and vk is a verification key of the signature scheme. The public-key of the IBE is a tuple (ek, vk) and the master secret-key is the signing key sk . A secret-key for an identity id is as short as a signature σ of id . At the time of encryption, we use ek to generate a pseudorandom value y corresponding to a statement $(\text{id}, v, \text{vk})$ with a witness u such that $\text{PRG}(\text{id} \oplus u) = v$. The ciphertext is a tuple (c_s, v) where c_s is a symmetric-key encryption (SKE) of a message m using y . Interestingly, the size of the ciphertext becomes optimal, that is $|m| + c\lambda$ where λ is a security parameter and c is a constant.

We need extractibility property of WPRF [33] to prove the security of IBE. However, we show (in Sect. 3) that the strong extractibility assumption can be avoided by replacing the normal signature scheme with a primitive called all-but-one signature (ABOS) [20]. We note that ABOS can be constructed from a verifiable random function (VRF) [26] and a perfectly-binding commitment scheme. Existing constructions [16, 25] of CCA secure IBE achieve (almost) optimal ciphertexts based on bilinear maps. Our result shows that assuming VRF

and a normal WPRF we can achieve a CCA secure IBE with optimal size ciphertexts. However, optimal ciphertext for IBE is not a primary contribution of this paper, rather we utilize our IBE to achieve more advanced primitive.

To convert the IBE into a MIFHE scheme (Sect. 3.1), we replace the SKE by a multi-key pure FHE which has been constructed using LWE assumption along with circular security [27]. In the pure MIFHE of [11] (based on obfuscation), the public-key of the underlying MFHE is unique for each identity, whereas there may be exponentially many MFHE public-keys associated to a single identity in our MIFHE and we have to include the MFHE public-key into a ciphertext so that evaluation runs smoothly. Therefore, MFHE is necessary for our construction even when messages are encrypted under the same identity.

2. Multi-attribute Pure FHE: To achieve a CCA1 secure MAFHE, we first realize a CCA secure attribute-based encryption (ABE) [32] using WPRF. Recall that a (key-policy) ABE enables us to encrypt messages under a set of attributes mapped to a bit-string x and a receiver holding a secret-key sk_f corresponding to a boolean function f should succeed in decrypting the ciphertext when x satisfies f . If we consider a WPRF for the language $L = \{(x, v, \text{vk}) : (\exists u \text{ such that } \text{PRG}(x \oplus u) = v) \text{ or } (\exists \sigma \text{ such that } \text{Vrfy}(\text{vk}, f, \sigma) = 1 \wedge f(x) = 1)\}$ similar to our basic IBE construction, then we can achieve a CCA secure ABE from OWFs. Here also we need to rely on extractability property of WPRF. To avoid this strong assumption, we start with the WE-based ABE of Garg et al. [15]. Specifically, the signature scheme is replaced with a witness-indistinguishable non-interactive zap [22]. The main difference from [15] is that to embed an attribute into a ciphertext we imitate the technique of embedding an identity from our IBE construction.

Goyal et al. [21] gave the first CCA secure ABE using bilinear maps. They used the generic technique of [2] to establish a bridge from CPA to CCA security for ABE. However, their transformation works in an environment where the CPA secure ABE has to support delegatability [21]. Another generic transformation was proposed in [32] which needs *verifiability* of a ciphertext encrypted under two different attributes. Our approach (in Sect. 4) defines a way to achieve a CCA secure ABE which is the first to enjoy an optimal ciphertext size (to the best of our knowledge).

We transform our ABE to a CCA1 secure MAFHE scheme (in Sect. 4.1) following the similar technique employed in the conversion of our MIFHE from the IBE. The MIFHEs and MAFHEs of [10, 11] are secure under the chosen-plaintext model which is often insufficient in many practical scenarios. Our approach leads to the first CCA1 secure MIFHE and CCA1 secure MAFHE without assuming $i\mathcal{O}$.

Other Related Works. Garg et al. [15] proposed constructions of IBE and ABE from witness encryption (WE) (introduced in the same work). Their selectively secure IBE is based on a dual encryption methodology and unique signature scheme. Replacing WE by WPRF does not immediately produce an optimal size ciphertext for the IBE. Using non-interactive zap and commitment schemes they built adaptively secure IBE and selectively secure ABE schemes. However, security holds in the CPA model and extension to MIFHE or MAFHE may require

additional primitive like obfuscation. Goldwasser et al. [19] built an ABE for Turing machines from WE and succinct argument of knowledge. But, their ABE is only CPA secure and based on strong extractibility assumptions.

2 Preliminaries

Notations. For any set S , the notation $x \leftarrow S$ denotes the process of sampling x uniformly at random from S . Let \mathcal{E} be a probabilistic polynomial time (PPT) algorithm. Then $y \leftarrow \mathcal{E}(x)$ denotes the execution of \mathcal{E} with an input x using fresh randomness and assign the output to y . If the randomness, say r , is provided externally then we denote this execution by $y \leftarrow \mathcal{E}(x; r)$. If $x \in \{0, 1\}^*$ then we denote by $|x|$ the size of x . We say $f : \mathbb{N} \rightarrow \mathbb{R}$ is a *negligible* function of n if it is $O(n^{-c})$ for all $c > 0$, and we use $\text{negl}(n)$ to denote a negligible function of n .

2.1 Pseudorandom Generator [1]

Definition 1. A pseudorandom generator (PRG) is a deterministic polynomial time algorithm PRG that on input a seed $s \in \{0, 1\}^\lambda$ outputs a string of length $\ell(\lambda)$ such that the following holds:

- *expansion*: For every λ it holds that $\ell(\lambda) > \lambda$.
- *pseudorandomness*: For all PPT adversary \mathcal{A} and $s \leftarrow \{0, 1\}^\lambda, r \leftarrow \{0, 1\}^{\ell(\lambda)}$, there exists a negligible function negl such that

$$\text{Adv}_{\mathcal{A}}^{\text{PRG}}(\lambda) = |\Pr[\mathcal{A}(1^\lambda, \text{PRG}(s)) = 1] - \Pr[\mathcal{A}(1^\lambda, r) = 1]| < \text{negl}(\lambda).$$

2.2 Symmetric Key Encryption [23, 24]

Definition 2. A symmetric key encryption (SKE) scheme is a tuple of PPT algorithms (Gen, Enc, Dec) defined as follows:

- $K \leftarrow \text{Gen}(1^\lambda)$: on input a security parameter λ , returns a key K .
- $c \leftarrow \text{Enc}(K, m)$: a randomized algorithm that returns c , an encryption of the message $m \in \mathcal{M}$.
- $\text{Dec}(K, c) \in \mathcal{M} \cup \{\perp\}$: a deterministic algorithm that decrypts the ciphertext c and returns a message $m \in \mathcal{M}$, or \perp if it fails.

The SKE is said to be correct if the following holds:

- *correctness*: For all $m \in \mathcal{M}$ and $K \leftarrow \text{Gen}(1^\lambda)$, we require that

$$\Pr[\text{Dec}(K, \text{Enc}(K, m)) = m] = 1$$

We consider chosen ciphertext attack (CCA) security for SKE and define an experiment $\text{Expt}_{\mathcal{A}, \text{CCA}}^{\text{SKE}}(1^\lambda)$ in Fig. 1.

$ \begin{aligned} &K \leftarrow \text{Gen}(1^\lambda) \\ &(m_0, m_1) \leftarrow \mathcal{A}^{\text{Enc}(K, \cdot), \text{Dec}(K, \cdot)}(1^\lambda) \\ &b \leftarrow \{0, 1\} \\ &c \leftarrow \text{Enc}(K, m_b) \\ &b' \leftarrow \mathcal{A}^{\text{Enc}(K, \cdot), \text{Dec}(K, \cdot)}(c) \\ &\text{if } (b' = b) \wedge (c \notin Q_K) \\ &\quad \text{return } 1 \\ &Q_K = \text{set of all Dec}(K, \cdot) \text{ queries} \end{aligned} $	$ \begin{aligned} &m^* \leftarrow \mathcal{A}(1^\lambda) \\ &(sk_0, vk_0) \leftarrow \text{Setup}(1^\lambda) \\ &(sk_1, vk_1) \leftarrow \text{PuncSetup}(1^\lambda, m^*) \\ &b \leftarrow \{0, 1\} \\ &b' \leftarrow \mathcal{A}^{\text{Sig}(sk_b, \cdot)}(vk_b) \\ &\text{if } (b = b') \wedge (m^* \notin Q_{sk}) \\ &\quad \text{return } 1 \\ &Q_{sk} = \text{set of all Sig}(sk_b, \cdot) \text{ queries} \end{aligned} $	$ \begin{aligned} &x^* \leftarrow \mathcal{A}(1^\lambda) \\ &(fk, ek) \leftarrow \text{Gen}(1^\lambda, R) \\ &y_0 \leftarrow F(fk, x^*), y_1 \leftarrow \mathcal{Y} \\ &b \leftarrow \{0, 1\} \\ &b' \leftarrow \mathcal{A}^{F(fk, \cdot)}(ek, y_b) \\ &\text{if } (b' = b) \wedge (x^* \notin L \cup Q_{fk}) \\ &\quad \text{return } 1 \\ &Q_{fk} = \text{set of all F}(fk, \cdot) \text{ queries} \end{aligned} $
---	--	---

Fig. 1. $\text{Expt}_{\mathcal{A}, \text{CCA}}^{\text{SKE}}(1^\lambda)$

Fig. 2. $\text{Expt}_{\mathcal{A}}^{\text{ABOS}}(1^\lambda)$

Fig. 3. $\text{Expt}_{\mathcal{A}}^{\text{WPRF}, R}(1^\lambda)$

Definition 3. A symmetric key encryption SKE is said to satisfy chosen ciphertext attack (CCA) security if, for all PPT adversary \mathcal{A} , there exists a negligible function negl such that

$$\text{Adv}_{\mathcal{A}, \text{CCA}}^{\text{SKE}}(\lambda) = |\Pr[\text{Expt}_{\mathcal{A}, \text{CCA}}^{\text{SKE}}(1^\lambda) = 1] - \frac{1}{2}| < \text{negl}(\lambda)$$

Remark 1. We take a *length preserving* SKE means $|\text{Enc}(K, m)| = |m|$. In such a scheme, \mathcal{A} is not allowed to query m_0 and m_1 to the encryption oracle. The CMC mode [23] and ECM mode [24], proposed by Halevi and Rogaway, is length preserving and CCA secure if the underlying block cipher is a strong pseudorandom permutation such as AES [14]. In fact, we need much weaker notion of CCA security where \mathcal{A} is not given the access of $\text{Enc}(K, \cdot)$. We term this notion as length preserving CCA (LP-CCA) secure SKE which is sufficient for our applications.

2.3 All-but-one Signature Scheme [20]

Definition 4. An all-but-one signature (ABOS) scheme is a tuple of PPT algorithms (Setup, PuncSetup, Sig, Vrfy) defined as follows:

- $(sk, vk) \leftarrow \text{Setup}(1^\lambda)$: on input a security parameter λ , outputs a signing key sk and a verification key vk .
- $(sk, vk) \leftarrow \text{PuncSetup}(1^\lambda, m^*)$: on input a security parameter λ and a message $m^* \in \mathcal{M}$, outputs a signing key sk and a verification key vk .
- $\sigma \leftarrow \text{Sig}(sk, m)$: returns $\sigma \in \Sigma$, a signature of the message $m \in \mathcal{M}$.
- $\text{Vrfy}(vk, m, \sigma) \in \{0, 1\}$: a deterministic algorithm that on input a verification key vk , a message m and a signature σ , and outputs either 0 or 1.

The signature scheme ABOS is said to be correct if the following holds:

- *correctness of Setup*: For all $m \in \mathcal{M}$ and $(sk, vk) \leftarrow \text{Setup}(1^\lambda)$, we require

$$\Pr[\text{Vrfy}(vk, m, \text{Sig}(sk, m)) = 1] = 1$$

- *correctness of PuncSetup*: For any $m^* \in \mathcal{M}$, $(sk, vk) \leftarrow \text{PuncSetup}(1^\lambda, m^*)$ and any $\sigma \in \Sigma$, we have $\text{Vrfy}(vk, m^*, \sigma) = 0$.

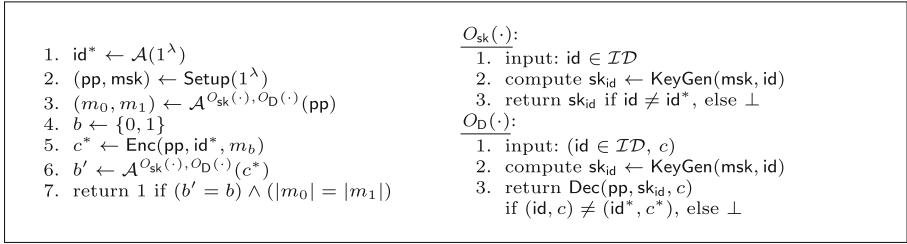


Fig. 4. $\text{Expt}_{\mathcal{A}, \text{CCA}}^{\text{IBE}}(1^\lambda)$

We consider VK indistinguishability experiment $\text{Expt}_{\mathcal{A}}^{\text{ABOS}}(1^\lambda)$ in Fig. 2.

Definition 5. An all-but-one signature ABOS scheme is said to satisfy VK indistinguishability (VK-IND) security if for all PPT adversary \mathcal{A} , there exists a negligible function negl such that

$$\text{Adv}_{\mathcal{A}}^{\text{ABOS}}(\lambda) = |\Pr[\text{Expt}_{\mathcal{A}}^{\text{ABOS}}(1^\lambda) = 1] - \frac{1}{2}| < \text{negl}(\lambda)$$

2.4 Witness Pseudorandom Function [33]

Definition 6. A witness pseudorandom function (WPRF) for an NP language L with a relation R is a tuple of PPT algorithms $(\text{Gen}, \text{F}, \text{Eval})$ defined as follows:

- $(\text{fk}, \text{ek}) \leftarrow \text{Gen}(1^\lambda, R)$: on input a security parameter λ and a relation circuit $R: \mathcal{X} \times \mathcal{W} \rightarrow \{0, 1\}$, returns a secret function key fk and a public evaluation key ek .
- $y \leftarrow \text{F}(\text{fk}, x)$: returns a pseudorandom value $y \in \mathcal{Y}$ for $x \in \mathcal{X}$.
- $\text{Eval}(\text{ek}, x, w) \in \mathcal{Y} \cup \{\perp\}$: on input an evaluation key ek , an element $x \in \mathcal{X}$ and a witness $w \in \mathcal{W}$, returns an element $y \in \mathcal{Y}$, or \perp if it fails.

We note that, each of the above algorithms except Gen is a deterministic algorithm. The WPRF is said to be correct if the following holds:

- *correctness of Eval*: For all $x \in \mathcal{X}, w \in \mathcal{W}$ and $(\text{fk}, \text{ek}) \leftarrow \text{Gen}(1^\lambda, R)$, we require that

$$\text{Eval}(\text{ek}, x, w) = \begin{cases} \text{F}(\text{fk}, x) & \text{if } R(x, w) = 1 \\ \perp & \text{if } R(x, w) = 0 \end{cases}$$

The security experiment $\text{Expt}_{\mathcal{A}}^{\text{WPRF}, R}(1^\lambda)$ for the WPRF is defined in Fig. 3. We consider a selective model which is sufficient for our applications.

Definition 7. A witness pseudorandom function WPRF for an NP language L with a relation R is said to be selectively secure if, for all PPT adversary \mathcal{A} , there exists a negligible function negl such that

$$\text{Adv}_{\mathcal{A}}^{\text{WPRF}, R}(\lambda) = |\Pr[\text{Expt}_{\mathcal{A}}^{\text{WPRF}, R}(1^\lambda) = 1] - \frac{1}{2}| < \text{negl}(\lambda)$$

3 CCA1 Secure MIFHE from WPRF and MFHE

The main building block of our MIFHE is a CCA secure IBE. Firstly, we use WPRF and ABOS to achieve a CCA secure IBE having an optimal size ciphertext. Then we extend it to a CCA1 secure MIFHE with the help of existing MFHE schemes. We begin with the definition of an IBE system.

Definition 8. [3] An identity-based encryption (IBE) scheme is a tuple of PPT algorithms (Setup, KeyGen, Enc, Dec) defined as follows:

- $(pp, msk) \leftarrow \text{Setup}(1^\lambda)$: on input a security parameter λ , produces a public parameter pp and a master secret-key msk .
- $sk_{id} \leftarrow \text{KeyGen}(msk, id)$: returns a secret-key sk_{id} corresponding to the identity $id \in \mathcal{ID}$ using a master secret-key msk .
- $c \leftarrow \text{Enc}(pp, id, m)$: returns c , an encryption of a message $m \in \mathcal{M}$ under an identity id .
- $\text{Dec}(pp, sk_{id}, c) \in \mathcal{M} \cup \{\perp\}$: a deterministic algorithm that decrypts a ciphertext c using a secret-key sk_{id} and outputs either a message $m \in \mathcal{M}$ or \perp if it fails.

The IBE is said to be correct if the following holds:

- *correctness*: For all $id \in \mathcal{ID}$, $m \in \mathcal{M}$, $(pp, msk) \leftarrow \text{Setup}(1^\lambda)$ and $sk_{id} \leftarrow \text{KeyGen}(msk, id)$, we require that

$$\Pr[\text{Dec}(pp, sk_{id}, \text{Enc}(pp, id, m)) = m] = 1$$

For security of IBE, we consider CCA security with selective-identity experiment $\text{Expt}_{\mathcal{A}, \text{CCA}}^{\text{IBE}}(1^\lambda)$ described in Fig. 4.

Definition 9. An identity-based encryption IBE is said to be selective-identity CCA secure if, for all PPT adversary \mathcal{A} , there exists a negligible function negl such that

$$\text{Adv}_{\mathcal{A}, \text{CCA}}^{\text{IBE}}(\lambda) = |\Pr[\text{Expt}_{\mathcal{A}, \text{CCA}}^{\text{IBE}}(1^\lambda) = 1] - \frac{1}{2}| < \text{negl}(\lambda)$$

Construction. We construct an identity-based encryption scheme $\text{IBE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$ for an identity space $\mathcal{ID} = \{0, 1\}^\lambda$. The following primitives are utilized:

- A pseudorandom generator $\text{PRG} : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$.
- A LP-CCA secure symmetric key encryption $\text{SKE} = (\text{Gen}, \text{Enc}, \text{Dec})$.
- A VK-IND secure all-but-one signature scheme $\text{ABOS} = (\text{Setup}, \text{PuncSetup}, \text{Sig}, \text{Vrfy})$ with the message space as \mathcal{ID} and signature space Σ .
- A WPRF $= (\text{Gen}, \text{F}, \text{Eval})$ for the NP language $L = \{(id, v, vk) : (\exists u \in \{0, 1\}^\lambda \text{ such that } \text{PRG}(id \oplus u) = v) \text{ or } (\exists \sigma \text{ such that } \text{ABOS.Vrfy}(vk, id, \sigma) = 1)\}$ with a relation $R : \mathcal{X} \times \mathcal{W} \rightarrow \{0, 1\}$. So, $R((id, v, vk), \omega) = 1$ if $(\text{PRG}(id \oplus \omega) = v) \vee (\text{Vrfy}(vk, id, \omega) = 1)$, 0 otherwise. Note that, we can always fix the input size of R by adding some dummy bits.

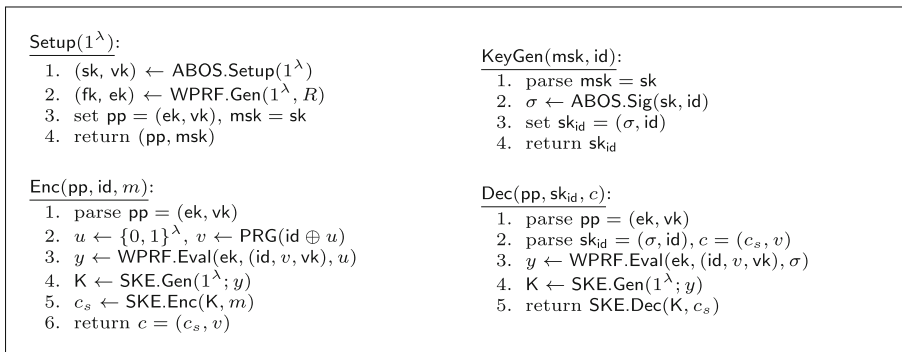


Fig. 5. Construction of IBE with optimal ciphertexts

We describe our IBE in Fig. 5. For *correctness*, we have to make sure that a same pseudorandom value y is generated in both the algorithms Enc and Dec. In Enc, we compute y using a witness u for PRG and in Dec, we compute y using a witness which is now a signature σ for id . More importantly, the statement (id, v, vk) remains unchanged in both cases. Thus, correctness of Eval ensures $y = \text{WPRF.F}(fk, (id, v, vk))$ is the same in Enc and Dec. Finally, Dec returns the message m using the decryption of SKE.

Efficiency: The ciphertext size of our IBE is compact in the sense that it has only $|c_s| + |v|$ many bits. Since c_s is a ciphertext of a length preserving SKE, we have $|c_s| = |m|$, where $|m|$ denotes the bit length of message. Therefore, the size of c is $|m| + 2\lambda$ which is *optimal* for any IBE scheme. The underlying relation R is also simple as it either checks a PRG or verify a message-signature pair. This means the size of public parameter is proportional to the size of PRG plus the size of Vrfy, hence is some fixed polynomial in λ .

Theorem 1. *The IBE = (Setup, KeyGen, Enc, Dec) described above is a selective-identity CCA secure identity based encryption if PRG is a secure pseudorandom generator, WPRF is a selectively secure witness pseudorandom function, ABOS is a VK-IND secure all-but-one signature scheme and SKE is a LP-CCA secure symmetric key encryption.*

Proof. We prove the security of IBE using the following sequence of games. As usual, we start with Game 0 which is the standard experiment $\text{Expt}_{\mathcal{A}}^{\text{IBE}}(\lambda)$ as defined in Fig. 4. For Game i , let G_i be the event $b = b'$. We assume that \mathcal{A} submits two messages of equal length in each game.

Game 0: This is the standard experiment as described in Definition 9. In particular, \mathcal{A} begins by committing to a challenge identity id^* . The challenger computes $(pp, msk) \leftarrow \text{Setup}(1^\lambda)$ and transfers pp to \mathcal{A} . The adversary, given access to the oracles $O_{sk}(\cdot), O_D(\cdot)$, submits a pair of challenge messages (m_0, m_1) . Next, the challenger chooses a random bit b and sends the

1. $id^* \leftarrow \mathcal{A}(1^\lambda)$
2. $(sk, vk) \leftarrow \text{ABOS.Setup}(1^\lambda)$
3. $(fk, ek) \leftarrow \text{WPRF.Gen}(1^\lambda, R)$
4. set $pp = (ek, vk)$, $msk = sk$
5. $(m_0, m_1) \leftarrow \mathcal{A}^{O_{sk}(\cdot), O_D(\cdot)}(pp)$
6. $u \leftarrow \{0, 1\}^\lambda$, $v \leftarrow \text{PRG}(id^* \oplus u)$
7. $y \leftarrow \text{WPRF.F}(fk, (id^*, v, vk))$
8. $K \leftarrow \text{SKE.Gen}(1^\lambda; y)$
9. $b \leftarrow \{0, 1\}$
10. $c_s^* \leftarrow \text{SKE.Enc}(K, m_b)$
11. set $c^* = (c_s^*, v)$
12. $b' \leftarrow \mathcal{A}^{O_{sk}(\cdot), O_D(\cdot)}(c^*)$
13. return 1 if $(b' = b)$

Fig. 6. Game 1

1. $id^* \leftarrow \mathcal{A}(1^\lambda)$
2. $(sk, vk) \leftarrow \text{ABOS.Setup}(1^\lambda)$
3. $(fk, ek) \leftarrow \text{WPRF.Gen}(1^\lambda, R)$
4. set $pp = (ek, vk)$, $msk = sk$
5. $(m_0, m_1) \leftarrow \mathcal{A}^{O_{sk}(\cdot), O_D(\cdot)}(pp)$
6. $v \leftarrow \{0, 1\}^{2\lambda}$
7. $y \leftarrow \text{WPRF.F}(fk, (id^*, v, vk))$
8. $K \leftarrow \text{SKE.Gen}(1^\lambda; y)$
9. $b \leftarrow \{0, 1\}$
10. $c_s^* \leftarrow \text{SKE.Enc}(K, m_b)$
11. set $c^* = (c_s^*, v)$
12. $b' \leftarrow \mathcal{A}^{O_{sk}(\cdot), O_D(\cdot)}(c^*)$
13. return 1 if $(b' = b)$

Fig. 7. Game 2

challenge ciphertext as $c^* \leftarrow \text{Enc}(pp, id^*, m_b)$. Finally, \mathcal{A} , given access to the same oracles, guesses the challenge bit b . Note that, \mathcal{A} cannot make a query id^* to $O_{sk}(\cdot)$ and a query (id^*, c^*) to $O_D(\cdot)$.

Game 1: It is same as Game 0 except that the challenger generates the randomness as $y \leftarrow \text{WPRF.F}(fk, (id^*, v, vk))$ instead of using Eval with the witness u . Game 1 is described in Fig. 6. It can be observed by the correctness of Eval

$$\text{WPRF.Eval}(ek, (id^*, v, vk), u) = \text{WPRF.F}(fk, (id^*, v, vk))$$

as $R((id^*, v, vk), u) = 1$. Therefore, the ciphertext distributions in games 0 and 1 are identical. This implies $\text{Pr}[G_0] = \text{Pr}[G_1]$.

Game 2: It is exactly same as Game 1 except that the challenger picks v uniformly at random from $\{0, 1\}^{2\lambda}$ instead of computing $v \leftarrow \text{PRG}(id^* \oplus u)$. Game 2 is described in Fig. 7. Since u is chosen uniformly at random from $\{0, 1\}^\lambda$, the distribution of $id^* \oplus u$ is also uniform over $\{0, 1\}^\lambda$. The security of PRG (Definition 1) ensures that \mathcal{A} 's advantage in distinguishing between Game 1 and Game 2 is $|\text{Pr}[G_1] - \text{Pr}[G_2]| = \text{Adv}_{\mathcal{B}_1}^{\text{PRG}}(\lambda) = \text{negl}(\lambda)$ where \mathcal{B}_1 is a PRG -adversary.

Game 3: It is similar to Game 2 except that the challenger computes $(sk^*, vk^*) \leftarrow \text{ABOS.PuncSetup}(1^\lambda, id^*)$ in the setup and replaces the key generation and decryption oracles with $O_{sk^*}(\cdot)$ and $O_{D, vk^*, K}(\cdot)$ respectively, defined in Fig. 8. Therefore, \mathcal{A} gets a public parameter of the form $pp = (ek, vk^*)$. In Lemma 1, we show that Game 2 and Game 3 are indistinguishable from \mathcal{A} 's view.

Game 4: It is identical to Game 3 except that the challenger selects y uniformly at random from \mathcal{Y} which is the co-domain of $\text{WPRF.F}(fk, \cdot)$ and replaces the decryption oracle $O_{D, vk^*, K}(\cdot)$ by $O_{D^*, vk^*, K}(\cdot)$, defined in Fig. 9. In Lemma 2, we show that Game 3 and Game 4 are indistinguishable from \mathcal{A} 's view.

Finally, we note that the encryption key in Game 4 is computed as $K \leftarrow \text{SKE.Gen}(1^\lambda; y)$ where y is a fresh randomness which is independent of the challenge identity id^* . Hence, by the LP-CCA security of SKE (Remark 1) we have

$|\Pr[G_4] - \frac{1}{2}| = \text{Adv}_{\mathcal{B}_2, \text{LP-CCA}}^{\text{SKE}}(\lambda)$ which is negligible in λ by our assumption. We are left to prove the following lemmas to conclude the security of our IBE.

Lemma 1. *Assuming ABOS is a VK-IND secure all-but-one signature scheme, we have $|\Pr[G_2] - \Pr[G_3]| = \text{negl}(\lambda)$.*

Proof. We show that if \mathcal{A} can distinguish between the games 2 and 3, then there exists an adversary \mathcal{B}_3 which will break the VK-IND security of ABOS (Definition 5). Let id^* be the challenge message for \mathcal{B}_3 which simulates \mathcal{A} as follows:
 $\mathcal{B}_3(1^\lambda, \text{id}^*)$:

1. send id^* to its challenger
2. ABOS-challenger does the following:
 - (a) $(\text{sk}_0, \text{vk}_0) \leftarrow \text{ABOS.Setup}(1^\lambda)$
 - (b) $(\text{sk}_1, \text{vk}_1) \leftarrow \text{ABOS.PuncSetup}(1^\lambda, m^*)$
 - (c) $\tilde{b} \leftarrow \{0, 1\}$
 - (d) return $\text{vk}_{\tilde{b}}$ to \mathcal{B}_3
3. generate $(\text{fk}, \text{ek}) \leftarrow \text{WPRF.Gen}(1^\lambda, R)$
4. pick $v \leftarrow \{0, 1\}^{2\lambda}$
5. set $y \leftarrow \text{WPRF.F}(\text{fk}, (\text{id}^*, v, \text{vk}_{\tilde{b}}))$
6. compute $\text{K} \leftarrow \text{SKE.Gen}(1^\lambda; y)$
7. set $\text{pp} = (\text{ek}, \text{vk}_{\tilde{b}})$ and send it to \mathcal{A}
8. \mathcal{A} can ask the following queries for polynomial number of times:
 - (a) *key query for id*: \mathcal{B}_3 uses its signing oracle $\text{ABOS.Sig}(\text{sk}_{\tilde{b}}, \cdot)$ to get a signature σ of id and return $\text{sk}_{\text{id}} = (\text{id}, \sigma)$ if $\text{id} \neq \text{id}^*$, else return \perp
 - (b) *ciphertext query for (id, c)*: \mathcal{B}_3 uses the function $O_{\text{D}, \text{vk}_{\tilde{b}}, \text{K}}(\cdot)$ defined in Fig. 8 for ciphertext query of \mathcal{A}
9. \mathcal{A} submits the challenge messages (m_0, m_1)
10. pick $b \leftarrow \{0, 1\}$ and computes $c_s^* \leftarrow \text{SKE.Enc}(\text{K}, m_b)$
11. set $c^* = (c_s^*, v)$ and send it to \mathcal{A}
12. \mathcal{A} may repeat the step 8 and returns a guess b' for b
13. return 1 if $b = b'$ and $|m_0| = |m_1|$

It is easy to see that if $\tilde{b} = 0$ then \mathcal{B}_3 simulates the KeyGen oracle $O_{\text{sk}}(\cdot)$ of Game 2 and if $\tilde{b} = 1$ then \mathcal{B}_3 simulates the KeyGen oracle $O_{\text{sk}^*}(\cdot)$ of Game 3. Next, we show that $O_{\text{D}, \text{vk}_0, \text{K}}(\cdot)$ works like the oracle $O_{\text{D}}(\cdot)$ as in Game 2. For any arbitrary query $(\text{id}, c = (\bar{c}_s, \bar{v}))$, let us consider the following cases.

Case 1 $(\text{id}, c) = (\text{id}^*, c^*)$: Both the oracles return \perp as it is not a valid query.

Case 2 $(\text{id}, \bar{v}) = (\text{id}^*, v) \wedge (\bar{c}_s \neq c_s^*)$: Let, $z_0 = (\text{id}^*, v, \text{vk}_0)$. The oracle $O_{\text{D}}(\cdot)$ generates a signature $\sigma \leftarrow \text{ABOS.Sign}(\text{sk}_0, \text{id}^*)$ (where $(\text{sk}_0, \text{vk}_0) \leftarrow \text{ABOS.Setup}(1^\lambda)$ as in Game 2, Fig. 7) and uses $y \leftarrow \text{WPRF.Eval}(\text{ek}, z_0, \sigma)$ to generate the decryption key. On the other hand, $O_{\text{D}, \text{vk}_0, \text{K}}(\cdot)$ uses $y^* \leftarrow \text{WPRF.F}(\text{fk}, z_0)$ to generate the decryption key. By the correctness of Eval , $y^* = y$ as $R(z_0, \sigma) = 1$.

Case 3 $(\text{id}, \bar{v}) \neq (\text{id}^*, v)$: Let $z = (\text{id}, \bar{v}, \text{vk}_0)$. The oracle $O_{\text{D}}(\cdot)$ generates a signature $\sigma \leftarrow \text{ABOS.Sig}(\text{sk}_0, \text{id})$ (as in Game 2) and uses $y \leftarrow \text{WPRF.Eval}(\text{ek}, z, \sigma)$

<ol style="list-style-type: none"> 1. $\text{id}^* \leftarrow \mathcal{A}(1^\lambda)$ 2. $(\text{sk}^*, \text{vk}^*) \leftarrow \text{ABOS.PuncSetup}(1^\lambda, \text{id}^*)$ 3. $(\text{fk}, \text{ek}) \leftarrow \text{WPRF.Gen}(1^\lambda, R)$ 4. set $\text{pp} = (\text{ek}, \text{vk}^*)$ and $\text{msk} = \text{sk}^*$ 5. $v \leftarrow \{0, 1\}^{2\lambda}$ 6. $y^* \leftarrow \text{WPRF.F}(\text{fk}, (\text{id}^*, v, \text{vk}^*))$ 7. $K \leftarrow \text{SKE.Gen}(1^\lambda; y^*)$ 8. $(m_0, m_1) \leftarrow \mathcal{A}^{O_{\text{sk}^*}(\cdot), O_{\text{D}, \text{vk}^*, K}(\cdot)}(\text{pp})$ 9. $b \leftarrow \{0, 1\}$ 10. $c_s^* \leftarrow \text{SKE.Enc}(K, m_b)$ 11. return $c^* = (c_s^*, v)$ 12. $b' \leftarrow \mathcal{A}^{O_{\text{sk}^*}(\cdot), O_{\text{D}, \text{vk}^*, K}(\cdot)}(c^*)$ 13. return 1 if $(b' = b)$ 	$O_{\text{sk}^*}(\cdot)$: <ol style="list-style-type: none"> 1. input: $\text{id} \in \mathcal{ID}$ 2. compute $\sigma \leftarrow \text{ABOS.Sig}(\text{sk}^*, \text{id})$ 3. return $\text{sk}_{\text{id}} = (\text{id}, \sigma)$ if $\text{id} \neq \text{id}^*$, else \perp $O_{\text{D}, \text{vk}^*, K}(\cdot)$: <ol style="list-style-type: none"> 1. input: $(\text{id} \in \mathcal{ID}, c)$ 2. parse $c = (\bar{c}_s, \bar{v})$ 3. if $(\text{id}, c) = (\text{id}^*, c^*)$ 4. return \perp 5. else if $(\text{id}, \bar{v}) = (\text{id}^*, v)$ 6. return $\text{SKE.Dec}(K, \bar{c}_s)$ 7. else $\bar{y} \leftarrow \text{WPRF.F}(\text{fk}, (\text{id}, \bar{v}, \text{vk}^*))$ 8. $\bar{K} \leftarrow \text{SKE.Gen}(1^\lambda; \bar{y})$ 9. return $\text{SKE.Dec}(\bar{K}, \bar{c}_s)$
---	--

Fig. 8. Game 3

to generate the decryption key. $O_{\text{D}, \text{vk}_0, K}(\cdot)$ uses $y \leftarrow \text{WPRF.F}(\text{fk}, z)$ to generate the decryption key. By the similar argument as in case 2, we conclude that both the oracles compute the same decryption key.

Thus, \mathcal{B}_3 perfectly simulates Game 2 when $\tilde{b} = 0$. On the other hand, when the ABOS challenger picks $\tilde{b} = 1$, it perfectly simulates Game 3. Therefore, the advantage of \mathcal{A} in distinguishing between the games 2 and 3 is the same as winning advantage of \mathcal{B}_3 in VK-IND security experiment and we write it as $|\Pr[\mathcal{G}_2] - \Pr[\mathcal{G}_3]| = \text{Adv}_{\mathcal{B}_3}^{\text{ABOS}}(\lambda)$ which is negligible in λ by our assumption.

Lemma 2. *Assuming WPRF is a selectively secure witness pseudorandom function, we have $|\Pr[\mathcal{G}_3] - \Pr[\mathcal{G}_4]| = \text{negl}(\lambda)$.*

Proof. We show that if \mathcal{A} can distinguish between the games 3 and 4, then there exists an adversary \mathcal{B}_4 which will break the selective security of WPRF (Definition 7). The challenge statement for \mathcal{B}_4 is $z^* = (\text{id}^*, v, \text{vk}^*)$ where $v \leftarrow \{0, 1\}^{2\lambda}$ and $(\text{sk}^*, \text{vk}^*) \leftarrow \text{ABOS.PuncSetup}(1^\lambda, \text{id}^*)$. Note that, $v \leftarrow \{0, 1\}^{2\lambda}$ implies that there exists $u \in \{0, 1\}^\lambda$ satisfying $\text{PRG}(\text{id}^* \oplus u) = v$ holds with a negligible probability of (at most) $2^{-\lambda}$. Furthermore, by the correctness of PuncSetup (Definition 4), we have $\text{ABOS.Vrfy}(\text{vk}^*, \text{id}^*, \sigma) = 0$ for all $\sigma \in \Sigma$. Hence, $R(z^*, w) = 0$ holds with overwhelming probability for any $w \in \mathcal{W}$ and z^* is a valid challenge statement for \mathcal{B}_4 . Below we describe how \mathcal{B}_4 simulates \mathcal{A} using z^* .

$\mathcal{B}_4(1^\lambda, z^*)$:

1. send z^* to its challenger
2. WPRF-challenger does the following:
 - (a) generate $(\text{fk}, \text{ek}) \leftarrow \text{WPRF.Gen}(1^\lambda, R)$
 - (b) set $y_0 \leftarrow \text{WPRF.F}(\text{fk}, z^*)$ and $y_1 \leftarrow \mathcal{Y}$
 - (c) pick $\tilde{b} \leftarrow \{0, 1\}$
 - (d) return $(\text{ek}, y_{\tilde{b}})$ to \mathcal{B}_4
3. compute $K \leftarrow \text{SKE.Gen}(1^\lambda; y_{\tilde{b}})$

4. set $\text{pp} = (\text{ek}, \text{vk}^*)$ and send it to \mathcal{A}
5. \mathcal{A} can query the following oracles for polynomial number of times:
 - (a) *key query for id*: \mathcal{B}_4 uses the oracle $O_{\text{sk}^*}(\cdot)$ as described in Fig. 9 to compute the secret-key for id
 - (b) *ciphertext query for (id, c)* : \mathcal{B}_4 uses the decryption oracle $O_{\text{D}^*, \text{vk}^*, \text{K}}(\cdot)$ as defined in Fig. 9 to compute the message for the query (id, c)
6. \mathcal{A} submits the challenge messages (m_0, m_1)
7. pick $b \leftarrow \{0, 1\}$ and computes $c_s^* \leftarrow \text{SKE.Enc}(\text{K}, m_b)$
8. set $c^* = (c_s^*, v)$ and send it to \mathcal{A}
9. \mathcal{A} may repeat the step 5 and returns a guess b' for b
10. return 1 if $b = b'$ and $|m_0| = |m_1|$

<ol style="list-style-type: none"> 1. $\text{id}^* \leftarrow \mathcal{A}(1^\lambda)$ 2. $(\text{sk}^*, \text{vk}^*) \leftarrow \text{ABOS.PuncSetup}(1^\lambda, \text{id}^*)$ 3. $(\text{fk}, \text{ek}) \leftarrow \text{WPRF.Gen}(1^\lambda, R)$ 4. set $\text{pp} = (\text{ek}, \text{vk}^*)$ and $\text{msk} = \text{sk}^*$ 5. $v \leftarrow \{0, 1\}^{2\lambda}$ 6. set $z^* = (\text{id}^*, v, \text{vk}^*)$ 7. $y \leftarrow \mathcal{Y}$ 8. $\text{K} \leftarrow \text{SKE.Gen}(1^\lambda; y)$ 9. $(m_0, m_1) \leftarrow \mathcal{A}^{O_{\text{sk}^*}(\cdot), O_{\text{D}^*, \text{vk}^*, \text{K}}(\cdot)}(\text{pp})$ 10. $b \leftarrow \{0, 1\}$ 11. $c_s^* \leftarrow \text{SKE.Enc}(\text{K}, m_b)$ 12. return $c^* = (c_s^*, v)$ 13. $b' \leftarrow \mathcal{A}^{O_{\text{sk}^*}(\cdot), O_{\text{D}^*, \text{vk}^*, \text{K}}(\cdot)}(c^*)$ 14. return 1 if $(b' = b)$ 	$O_{\text{sk}^*}(\cdot)$: <ol style="list-style-type: none"> 1. input: $\text{id} \in \mathcal{ID}$ 2. compute $\sigma \leftarrow \text{ABOS.Sig}(\text{sk}^*, \text{id})$ 3. return $\text{sk}_{\text{id}} = (\text{id}, \sigma)$ if $\text{id} \neq \text{id}^*$, else \perp $O_{\text{D}^*, \text{vk}^*, \text{K}}(\cdot)$: <ol style="list-style-type: none"> 1. input: $(\text{id} \in \mathcal{ID}, c)$ 2. parse $c = (\bar{c}_s, \bar{v})$ 3. if $(\text{id}, c) = (\text{id}^*, c^*)$ 4. return \perp 5. else if $(\text{id}, \bar{v}) = (\text{id}^*, v)$ 6. return $\text{SKE.Dec}(\text{K}, \bar{c}_s)$ 7. else $\bar{y} \leftarrow O_{\text{fk}}((\text{id}, \bar{v}, \text{vk}^*))$ 8. $\bar{\text{K}} \leftarrow \text{SKE.Gen}(1^\lambda; \bar{y})$ 9. return $\text{SKE.Dec}(\bar{\text{K}}, \bar{c}_s)$ <p style="margin-left: 20px;">Here $O_{\text{fk}}(z) = \text{WPRF.F}(\text{fk}, z)$ if $z \neq z^*$, else \perp</p>
---	--

Fig. 9. Game 4

First, we note that the oracle $O_{\text{sk}^*}(\cdot)$ remains the same as in Game 3. Next, we observe that if $\tilde{b} = 0$ then the decryption oracles $O_{\text{D}, \text{vk}^*, \text{K}}(\cdot)$ of Game 3 and $O_{\text{D}^*, \text{vk}^*, \text{K}}(\cdot)$ of Game 4 are functionally equivalent. More precisely, for any arbitrary query $(\text{id}, c = (\bar{c}_s, \bar{v}))$ we consider the following cases.

Case 1 $(\text{id}, c) = (\text{id}^*, c^*)$: Both the oracles return \perp as it is not a valid query.

Case 2 $(\text{id}, \bar{v}) = (\text{id}^*, v) \wedge (\bar{c}_s \neq c_s^*)$: Both the oracles $O_{\text{D}, \text{vk}^*, \text{K}}(\cdot)$ and $O_{\text{D}^*, \text{vk}^*, \text{K}}(\cdot)$ utilize $y_0 \leftarrow \text{WPRF.F}(\text{fk}, z^*)$ to generate the decryption key.

Case 3 $(\text{id}, \bar{v}) \neq (\text{id}^*, v)$: Let $z = (\text{id}, \bar{v}, \text{vk}^*) \neq z^*$. Then, $O_{\text{D}, \text{vk}^*, \text{K}}(\cdot)$ computes $y \leftarrow \text{WPRF.F}(\text{fk}, z)$ to generate the decryption key. On the other hand, $O_{\text{D}^*, \text{vk}^*, \text{K}}(\cdot)$ uses $y \leftarrow O_{\text{fk}}(z)$ to generate the decryption key. Note that $O_{\text{fk}}(z) = \text{WPRF.F}(\text{fk}, z)$ as $z \neq z^*$. Hence, both oracles compute the same decryption key. Therefore, if the WPRF challenger picks the bit $\tilde{b} = 0$, then $y_{\tilde{b}} = \text{WPRF.F}(\text{fk}, (\text{id}^*, v, \text{vk}^*))$ and hence \mathcal{B}_4 simulates Game 3. If $\tilde{b} = 1$ then y is chosen uniformly at random from \mathcal{Y} and hence \mathcal{B}_4 simulates Game 4. This implies

that the advantage of \mathcal{A} in distinguishing between the games 3 and 4 is the same as the advantage of \mathcal{B}_4 in the WPRF security experiment. Therefore, $|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_4]| = \text{Adv}_{\mathcal{B}_4}^{\text{WPRF},R}(\lambda)$ which is negligible in λ by our assumption.

3.1 From IBE to CCA1 Secure MIFHE

In this section, we describe our transformation from the above IBE to MIFHE. At first, we recall the definition of MFHE given by Mukherjee and Wichs [27] where they built a (pure) MFHE based on LWE along with circular security.

Definition 10. [27] A multi-key (pure) fully homomorphic encryption (MFHE) scheme is a tuple of PPT algorithms (Setup, KeyGen, Enc, Expand, Eval, Dec) defined as follows:

- $\text{params} \leftarrow \text{Setup}(1^\lambda)$: on input a security parameter λ , produces a system parameter params (which implicitly available to all other algorithms).
- $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{params})$: on input a system parameter params , outputs a secret-key sk and a public-key pk .
- $c \leftarrow \text{Enc}(\text{pk}, m)$: returns c , a *fresh* ciphertext for a message $m \in \{0, 1\}$.
- $\widehat{c} \leftarrow \text{Expand}((\text{pk}_1, \dots, \text{pk}_N), i, c)$: a deterministic algorithm that on input a sequence of N public-keys $(\text{pk}_1, \dots, \text{pk}_N)$ and a fresh ciphertext c encrypted under the i^{th} key pk_i , returns an *expanded* ciphertext \widehat{c} .
- $\widehat{c} \leftarrow \text{Eval}(\text{params}, C, (\widehat{c}_1, \dots, \widehat{c}_\ell))$: a deterministic algorithm that on input a polynomial-size boolean circuit C and a sequence of ℓ expanded ciphertexts $(\widehat{c}_1, \dots, \widehat{c}_\ell)$, outputs an *evaluated* ciphertext \widehat{c} .
- $\text{Dec}(\text{params}, (\text{sk}_1, \dots, \text{sk}_N), c) \in \{0, 1\} \cup \{\perp\}$: a deterministic algorithm that on input N secret-keys $\text{sk}_1, \dots, \text{sk}_N$ and a ciphertext c , returns either a message $m \in \{0, 1\}$ or \perp if it fails.

The MFHE is said to be correct and compact if the following holds:

For $\text{params} \leftarrow \text{Setup}(1^\lambda)$, $\{(\text{pk}_i, \text{sk}_i) \leftarrow \text{KeyGen}(\text{params})\}_{i \in [N]}$ and any ℓ -tuple message $(m_1, \dots, m_\ell) \in \{0, 1\}^\ell$, any sequence of indices $(I_1, \dots, I_\ell) \in [N]^\ell$, $\{c_i \leftarrow \text{Enc}(\text{pk}_{I_i}, m_i)\}_{i \in [\ell]}$, $\{\widehat{c}_i \leftarrow \text{Expand}((\text{pk}_1, \dots, \text{pk}_N), I_i, c_i)\}_{i \in [\ell]}$ and a polynomial-size boolean circuit C , we have

- *correctness of Expand*: $\text{Dec}(\text{params}, (\text{sk}_1, \dots, \text{sk}_N), \widehat{c}_i) = m_i$ for all $i \in [\ell]$.
- *correctness of Eval*: $\text{Dec}(\text{params}, (\text{sk}_1, \dots, \text{sk}_N), \widehat{c}) = C(m_1, \dots, m_\ell)$ where $\widehat{c} \leftarrow \text{Eval}(\text{params}, C, (\widehat{c}_1, \dots, \widehat{c}_\ell))$.
- *compactness*: The size of an evaluated ciphertext $|\widehat{c}|$ is bounded by a fixed polynomial $p(\lambda, N)$ independent of the circuit C .

Definition 11. A MFHE scheme is said to be semantically secure if, for all PPT adversary \mathcal{A} and $\text{params} \leftarrow \text{Setup}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{params})$, any pair of messages $(m_0, m_1) \in \{0, 1\}^2$, there exists a negligible function negl such that

$$\text{Adv}_{\mathcal{A}}^{\text{MFHE}}(\lambda) = |\Pr[\mathcal{A}(\text{params}, \text{pk}, \text{Enc}(\text{pk}, m_0)) = 1] - \Pr[\mathcal{A}(\text{params}, \text{pk}, \text{Enc}(\text{pk}, m_1)) = 1]| < \text{negl}(\lambda)$$

Definition 12. [8] A multi-identity (pure) fully homomorphic encryption (MIFHE) scheme is a tuple of PPT algorithms (Setup, KeyGen, Enc, Eval, Dec) where Setup, KeyGen and Enc are the same as in a normal IBE scheme (Definition 8) and the remaining two algorithms work as follows:

- $\hat{c} \leftarrow \text{Eval}(\text{pp}, C, (c_1, \dots, c_\ell))$: a deterministic algorithm that on input a public parameter pp , a polynomial-size boolean circuit C and ciphertexts c_1, \dots, c_ℓ (each of which encrypts a bit using Enc), outputs an evaluated ciphertext \hat{c} .
- $\text{Dec}(\text{pp}, (\text{sk}_{\text{id}_1}, \dots, \text{sk}_{\text{id}_\ell}), c) \in \{0, 1\} \cup \{\perp\}$: a deterministic algorithm that on input a public parameter pp , ℓ secret-keys $\text{sk}_{\text{id}_1}, \dots, \text{sk}_{\text{id}_\ell}$ corresponding to the identities $\text{id}_1, \dots, \text{id}_\ell$ and a ciphertext c encrypted under the identities $\text{id}_1, \dots, \text{id}_\ell$, outputs either a message $m \in \{0, 1\}$ or \perp if it fails.

The MIFHE is said to be correct and compact if the following hold:

- *correctness*: For $(\text{pp}, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$, $\{\text{sk}_{\text{id}_i} \leftarrow \text{KeyGen}(\text{msk}, \text{id}_i)\}_{i \in [\ell]}$ and any ℓ -tuple message $(m_1, \dots, m_\ell) \in \{0, 1\}^\ell$ such that $\{c_i \leftarrow \text{Enc}(\text{pp}, \text{id}_i, m_i)\}_{i \in [\ell]}$ and a polynomial-size boolean circuit C , we have

$$\Pr[\text{Dec}(\text{pp}, (\text{sk}_{\text{id}_1}, \dots, \text{sk}_{\text{id}_\ell}), \text{Eval}(\text{pp}, C, (c_1, \dots, c_\ell))) = C(m_1, \dots, m_\ell)] = 1$$

- *compactness*: The size of an evaluated ciphertext $|\hat{c}|$ is bounded by a fixed polynomial $p(\lambda, N)$ independent of the circuit C .

We consider CCA1 security for MIFHE where the adversary has an access to the decryption oracle before it receives the challenge ciphertext. We skip the formal description of the security as it is almost similar to Definition 9.

Construction. We construct a multi-identity pure FHE scheme MIFHE = (Setup, KeyGen, Enc, Eval, Dec) for an identity space $\mathcal{ID} = \{0, 1\}^\lambda$, a message space $\{0, 1\}$ and a class of polynomial sized circuits $\{\mathcal{C}_\lambda\}$. We consider the same set of primitives that are employed in the basic IBE of Sect. 3 except SKE is replaced by a pure MFHE scheme. Our MIFHE is described in Fig. 10. The correctness is followed by a similar argument as in our IBE scheme and using the correctness of MFHE scheme. We state the security in the following theorem.

Theorem 2. *The MIFHE = (Setup, KeyGen, Enc, Eval, Dec) described in Fig. 10 is a selective-identity CCA1 secure multi-identity pure fully homomorphic encryption if PRG is a secure pseudorandom generator, WPRF is a selectively secure puncturable witness pseudorandom function, ABOS is a VK-IND secure all-but-one signature scheme and MFHE is a semantically secure multi-key pure fully homomorphic encryption.*

Proof. The proof is similar to the Theorem 1 with few changes. Firstly, we replace SKE with MFHE. Secondly, observe that the semantic security of MFHE is sufficient as we consider CCA1 security for which \mathcal{A} is not allowed to query

<p><u>Setup(1^λ):</u></p> <ol style="list-style-type: none"> 1. $(sk, vk) \leftarrow \text{ABOS.Setup}(1^\lambda)$ 2. $(fk, ek) \leftarrow \text{WPRF.Gen}(1^\lambda, R)$ 3. $\text{params} \leftarrow \text{MFHE.Setup}(1^\lambda)$ 4. set $pp = (ek, vk, \text{params})$, $\text{msk} = sk$ 5. return (pp, msk) <p><u>KeyGen(msk, id):</u></p> <ol style="list-style-type: none"> 1. parse $\text{msk} = sk$ 2. $\sigma \leftarrow \text{ABOS.Sig}(sk, \text{id})$ 3. set $sk_{\text{id}} = (\sigma, \text{id})$ 4. return sk_{id} <p><u>Eval($pp, C, (\text{ct}_1, \dots, \text{ct}_\ell)$):</u></p> <ol style="list-style-type: none"> 1. parse $pp = (ek, vk, \text{params})$ 2. parse $\text{ct}_i = (c_{v_i}, v_i, \text{pk}_{v_i}), \forall i \in [\ell]$ 3. for $i = 1$ to ℓ 4. $\widehat{c}_i \leftarrow \text{MFHE.Expand}((\text{pk}_{v_1}, \dots, \text{pk}_{v_\ell}), i, c_{v_i})$ 5. $\widehat{c} \leftarrow \text{MFHE.Eval}(\text{params}, C, (\widehat{c}_1, \dots, \widehat{c}_\ell))$ 6. return $\widehat{ct} = (\widehat{c}, \{v_i, \text{pk}_{v_i}\}_{i \in [\ell]})$ 	<p><u>Enc(pp, id, m):</u></p> <ol style="list-style-type: none"> 1. parse $pp = (ek, vk, \text{params})$ 2. $u \leftarrow \{0, 1\}^\lambda, v \leftarrow \text{PRG}(\text{id} \oplus u)$ 3. $y_v \leftarrow \text{WPRF.Eval}(ek, (\text{id}, v, vk), u)$ 4. $(\text{pk}_v, \text{sk}_v) \leftarrow \text{MFHE.KeyGen}(\text{params}; y_v)$ 5. $c_v \leftarrow \text{MFHE.Enc}(\text{pk}_v, m)$ 6. return $\text{ct} = (c_v, v, \text{pk}_v)$ <p><u>Dec($pp, (sk_{\text{id}_1}, \dots, sk_{\text{id}_\ell}), \widehat{ct}$):</u></p> <ol style="list-style-type: none"> 1. parse $pp = (ek, vk, \text{params})$ 2. parse $sk_{\text{id}_i} = (\sigma_i, \text{id}_i), \forall i \in [\ell]$ 3. parse $\widehat{ct} = (\widehat{c}, \{v_i, \text{pk}_{v_i}\}_{i \in [\ell]})$ 4. for $i = 1$ to ℓ 5. $y_i \leftarrow \text{WPRF.Eval}(ek, (\text{id}_i, v_i, vk), \sigma_i)$ 6. $(\text{pk}_i, \text{sk}_i) \leftarrow \text{MFHE.KeyGen}(\text{params}; y_i)$ 7. if $\text{pk}_i \neq \text{pk}_{v_i}$ 8. return \perp 9. return $\text{MFHE.Dec}(\text{params}, (sk_1, \dots, sk_\ell), \widehat{c})$
---	--

Fig. 10. Construction of multi-identity pure FHE

the decryption oracle after the challenge query. More specifically, the secret-key sk_v , associated with the public-key pk_v which encrypts the challenge message, is no longer needed for any decryption oracle used in the proof. This is due to the fact that after Game 2 the component v of the challenge ciphertext (c_v, v, pk_v) is chosen uniformly from $\{0, 1\}^{2\lambda}$ and hence for all the decryption queries $\{(\text{id}, (\bar{c}_v, \bar{v}, \bar{\text{pk}}_v))\}$ of \mathcal{A} we have $v \neq \bar{v}$ with overwhelming probability. Thus, we omit the lines 5 and 6 from both the oracles $O_{D, \text{vk}^*, \text{K}}$ and $O_{D^*, \text{vk}^*, \text{K}}$, and rename them by O_{D, vk^*} and O_{D^*, vk^*} respectively. Finally, at the end of Game 4 we generate the key pair $(\text{pk}_v, \text{sk}_v) \leftarrow \text{MFHE.KeyGen}(\text{params}; y_v)$ using a fresh randomness y_v which is independent of the challenge identity id^* . Therefore, the semantic security of MFHE guarantees that $(\text{MFHE.Enc}(\text{pk}_v, 0), v, \text{pk}_v)$ is indistinguishable from $(\text{MFHE.Enc}(\text{pk}_v, 1), v, \text{pk}_v)$ which completes the proof.

4 CCA1 Secure MAFHE from WPRF and MFHE

In this section, we present a construction of a CCA1 secure multi-attribute pure FHE (MAFHE) using WPRF and MFHE. The heart of our MAFHE is a CCA secure (key-policy) ABE. We start with the definition of ABE.

Definition 13. [32] An attribute-based encryption (ABE) scheme for a class of functions $\{\mathcal{F}_\lambda\}$ is a tuple of PPT algorithms (Setup, KeyGen, Enc, Dec) defined as follows:

- $(pp, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$: on input a security parameter λ , produces a public parameter pp and a master secret-key msk .
- $sk_f \leftarrow \text{KeyGen}(pp, \text{msk}, f)$: returns a secret-key sk_f corresponding to the function $f \in \mathcal{F}_\lambda$.

- $c \leftarrow \text{Enc}(\text{pp}, x, m)$: returns c , an encryption of a message $m \in \mathcal{M}$ under an attribute $x \in \mathcal{X}$.
- $\text{Dec}(\text{pp}, \text{sk}_f, c) \in \mathcal{M} \cup \{\perp\}$: a deterministic algorithm that decrypts a ciphertext c using sk_f and outputs either a message $m \in \mathcal{M}$ or \perp if it fails.

The ABE is said to be correct if the following holds:

- *correctness*: For all $f \in \mathcal{F}_\lambda$, $x \in \mathcal{X}$, $m \in \mathcal{M}$, $(\text{pp}, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$ and $\text{sk}_f \leftarrow \text{KeyGen}(\text{msk}, \text{id})$, we require that

$$\Pr[\text{Dec}(\text{pp}, \text{sk}_f, \text{Enc}(\text{pp}, x, m)) = m : f(x) = 1] = 1$$

We consider selective-attribute CCA security for ABE and define the security experiment $\text{Expt}_{\mathcal{A}, \text{CCA}}^{\text{ABE}}(1^\lambda)$ in Fig. 11.

<ol style="list-style-type: none"> 1. $x^* \leftarrow \mathcal{A}(1^\lambda)$ 2. $(\text{pp}, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$ 3. $(m_0, m_1) \leftarrow \mathcal{A}^{O_{\text{sk}(\cdot)}, O_{\text{D}(\cdot)}}(\text{pp})$ 4. $b \leftarrow \{0, 1\}$ 5. $c^* \leftarrow \text{Enc}(\text{pp}, x^*, m_b)$ 6. $b' \leftarrow \mathcal{A}^{O_{\text{sk}(\cdot)}, O_{\text{D}(\cdot)}}(c^*)$ 7. return 1 if $(b' = b) \wedge (m_0 = m_1)$ 	$O_{\text{sk}(\cdot)}$: <ol style="list-style-type: none"> 1. input: $f \in \mathcal{F}_\lambda$ 2. compute $\text{sk}_f \leftarrow \text{KeyGen}(\text{msk}, f)$ 3. return sk_f if $f(x^*) = 0$, else \perp $O_{\text{D}(\cdot)}$: <ol style="list-style-type: none"> 1. input: $(f \in \mathcal{F}_\lambda, c)$ 2. compute $\text{sk}_f \leftarrow \text{KeyGen}(\text{msk}, f)$ 3. return $\text{Dec}(\text{pp}, \text{sk}_f, c)$ unless $(f, c) = (f, c^*) \wedge f(x^*) = 1$, else \perp
---	--

Fig. 11. $\text{Expt}_{\mathcal{A}, \text{CCA}}^{\text{ABE}}(1^\lambda)$

Definition 14. An attribute-based encryption ABE is said to be selective-attribute CCA secure if, for all PPT adversary \mathcal{A} , there exists a negligible function negl such that

$$\text{Adv}_{\mathcal{A}, \text{CCA}}^{\text{ABE}}(\lambda) = |\Pr[\text{Expt}_{\mathcal{A}, \text{CCA}}^{\text{ABE}}(1^\lambda) = 1] - \frac{1}{2}| < \text{negl}(\lambda)$$

Construction. We construct a selective-attribute CCA secure ABE based on the ABE of [15] which was built using witness encryption. The following ingredients are utilized:

- A pseudorandom generator $\text{PRG} : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$.
- A LP-CCA secure symmetric key encryption $\text{SKE} = (\text{Gen}, \text{Enc}, \text{Dec})$.
- A perfectly binding and computationally hiding commitment scheme $\text{Com}(\cdot)$.
- A non-interactive $\text{zap} = (\text{Prv}, \text{Vrfy})$ for the NP language $L' = \{(\eta_1, \eta_2, f) : (\exists w_1 \text{ such that } \eta_1 = \text{Com}(0; w_1)) \text{ or } (\exists (w_2, x) \text{ such that } \eta_2 = \text{Com}(0^n; w_2) \wedge f(x) = 0)\}$.
- A WPRF = $(\text{Gen}, \text{F}, \text{Eval})$ for the NP language $L = \{(x, v) : (\exists u \in \{0, 1\}^\lambda \text{ such that } \text{PRG}(x \oplus u) = v) \text{ or } (\exists (\eta_1, \eta_2, f, \pi) \text{ such that } \text{Vrfy}((\eta_1, \eta_2, f), \pi) = 1 \wedge f(x) = 1)\}$ with a relation $R : \mathcal{X} \times \mathcal{W} \rightarrow \{0, 1\}$.

<p><u>Setup</u>(1^λ):</p> <ol style="list-style-type: none"> 1. $(fk, ek) \leftarrow \text{WPRF.Gen}(1^\lambda, R)$ 2. $\eta_1 = \text{Com}(0; r), \eta_2 = \text{Com}(0^\lambda; s)$ 3. set $pp = (ek, \eta_1, \eta_2), msk = r$ 4. return (pp, msk) 	<p><u>KeyGen</u>(pp, msk, id):</p> <ol style="list-style-type: none"> 1. parse $pp = (ek, \eta_1, \eta_2), msk = r$ 2. $\pi_f \leftarrow \text{zap.Priv}((\eta_1, \eta_2, f), r)$ 3. set $sk_f = (f, \pi_f)$ 4. return sk_f
<p><u>Enc</u>(pp, x, m):</p> <ol style="list-style-type: none"> 1. parse $pp = (ek, \eta_1, \eta_2)$ 2. $u \leftarrow \{0, 1\}^\lambda, v \leftarrow \text{PRG}(x \oplus u)$ 3. $y \leftarrow \text{WPRF.Eval}(ek, (x, v), u)$ 4. $K \leftarrow \text{SKE.Gen}(1^\lambda; y)$ 5. $c_x \leftarrow \text{SKE.Enc}(K, m)$ 6. return $c = (x, c_x, v)$ 	<p><u>Dec</u>(pp, sk_f, c):</p> <ol style="list-style-type: none"> 1. parse $pp = (ek, \eta_1, \eta_2)$ 2. parse $sk_f = (f, \pi), c = (x, \hat{c}, v)$ 3. $y \leftarrow \text{WPRF.Eval}(ek, (x, v), (\eta_1, \eta_2, f, \pi))$ 4. $K \leftarrow \text{SKE.Gen}(1^\lambda; y)$ 5. return $\text{SKE.Dec}(K, \hat{c})$

Fig. 12. Construction of ABE with optimal ciphertexts

We describe our construction in Fig. 12. For *correctness*, we notice that whenever $f(x) = 1$ holds (η_1, η_2, f, π) becomes a valid witness of the statement (x, v) corresponding to the relation R of WPRF where $\pi \leftarrow \text{zap.Priv}((\eta_1, \eta_2, f), r)$. In other words, $\text{zap.Vrfy}((\eta_1, \eta_2, f), \pi) = 1$ and we have

$$\begin{aligned} \text{WPRF.F}(fk, (x, v)) &= \text{WPRF.Eval}(ek, (x, v), (\eta_1, \eta_2, f, \pi)) && \text{[Decryption]} \\ &= \text{WPRF.Eval}(ek, (x, v), u) && \text{[Encryption]} \end{aligned}$$

Therefore, the same randomness is used to obtain the SKE key during encryption and decryption if $f(x) = 1$ and the original message can be recovered from \hat{c} . The key efficiency factor is that the size of ciphertext (excluding the size of the attribute) is $|c| = |c_x| + |v| = |m| + 2\lambda$ which is *optimal* for any ABE scheme. Note that, plaintext and ciphertext sizes are the same for the SKE encryption.

Theorem 3. *The ABE = (Setup, KeyGen, Enc, Dec) described in Fig. 12 is a selective-attribute CCA secure attribute-based encryption if PRG is a secure pseudorandom generator, Com is a perfectly binding and computationally hiding commitment scheme, zap is a non-interactive zap, WPRF is a selectively secure puncturable witness pseudorandom function and SKE is a LP-CCA secure symmetric key encryption. (The proof is available in the full version.)*

4.1 From ABE to CCA1 Secure MAFHE

This section is devoted to present a CCA1 secure multi-attribute pure FHE (MAFHE) using the technique involved in our ABE and a multi-key pure FHE. At first, we state a formal definition of MAFHE.

Definition 15. A multi-attribute (pure) fully homomorphic encryption (MAFHE) scheme for a function class $\{\mathcal{F}_\lambda\}$ and an attribute space \mathcal{X} is a tuple of PPT algorithms (Setup, KeyGen, Enc, Eval, Dec) where Setup, KeyGen and Enc are the same as in a normal ABE scheme (Definition 13). The remaining two algorithms work as follows:

- $\hat{c} \leftarrow \text{Eval}(\text{pp}, C, (c_1, \dots, c_\ell))$: a deterministic algorithm that on input a public parameter pp , a boolean circuit C of polynomial size and ciphertexts c_1, \dots, c_ℓ (each of which encrypts a bit using Enc), outputs an evaluated ciphertext \hat{c} .
- $\text{Dec}(\text{pp}, (\text{sk}_{f_1}, \dots, \text{sk}_{f_\ell}), c) \in \{0, 1\} \cup \{\perp\}$: a deterministic algorithm that on input a public parameter pp , a sequence of secret-keys $(\text{sk}_{f_1}, \dots, \text{sk}_{f_\ell})$ corresponding to the functions $f_1, \dots, f_\ell \in \mathcal{F}_\lambda$ and a ciphertext c encrypted under the attributes $x_1, \dots, x_\ell \in \mathcal{X}$, outputs either a message $m \in \{0, 1\}$ or \perp if it fails.

The MAFHE is said to be correct and compact if the following hold:

- *correctness*: For $(\text{pp}, \text{msk}) \leftarrow \text{Setup}(1^\lambda)$, $\{\text{sk}_{f_i} \leftarrow \text{KeyGen}(\text{pp}, \text{msk}, f_i)\}_{i \in [\ell]}$ and any ℓ -tuple messages $(m_1, \dots, m_\ell) \in \{0, 1\}^\ell$ such that $\{c_i \leftarrow \text{Enc}(\text{pp}, x_i, m_i)\}_{i \in [\ell]}$ satisfying $f_i(x_i) = 1 \forall i \in [\ell]$ and a boolean circuit C of polynomial size, we have

$$\Pr[\text{Dec}(\text{pp}, (\text{sk}_{f_1}, \dots, \text{sk}_{f_\ell}), \text{Eval}(\text{pp}, C, (c_1, \dots, c_\ell))) = C(m_1, \dots, m_\ell)] = 1$$

- *compactness*: There exists a fixed polynomial $p(\cdot)$ such that the size of an evaluated ciphertext is bounded by $p(\lambda)$. This means $|\hat{c}|$ does not depend on the circuit C .

We consider CCA1 security for MAFHE where the adversary is given access to the decryption oracle until it receives the challenge ciphertext. We skip the formal description of the security as it is almost similar to Definition 14 where the decryption oracle is not provided after generating the challenge ciphertext.

Construction. We are all set to describe a MAFHE scheme based on our ABE. The idea is similar to how we built the MIFHE from our IBE. Consequently, we need the same set of primitives as required in the ABE of Sect. 4 except the SKE is replaced by a semantically secure pure MFHE. The MAFHE for a function class $\{\mathcal{F}_\lambda\}$ and message space $\{0, 1\}$ is described in Fig. 13. Note that, the setup algorithm does not take into account any predefined depth of supported circuits as we assume circular security of the underlying MFHE. The correctness can be similarly argued as in our ABE scheme along with the correctness of MFHE. The CCA1 security of our MAFHE is followed from the proof of Theorem 3.

Theorem 4. *The MAFHE = (Setup, KeyGen, Enc, Eval, Dec) described in Fig. 13 is a selective-attribute CCA1 secure multi-attribute pure fully homomorphic encryption if PRG is a secure pseudorandom generator, Com is a perfectly binding and computationally hiding commitment scheme, zap is a non-interactive zap, WPRF is a selectively secure puncturable witness pseudorandom function and MFHE is a semantically secure multi-key pure fully homomorphic encryption. (The proof is discussed in the full version.)*

<p>Setup(1^λ):</p> <ol style="list-style-type: none"> 1. $(fk, ek) \leftarrow \text{WPRF.Gen}(1^\lambda, R)$ 2. $\eta_1 = \text{Com}(0; r), \eta_2 = \text{Com}(0^\lambda; s)$ 3. $\text{params} \leftarrow \text{MFHE.Setup}(1^\lambda)$ 4. set $\text{pp} = (ek, \eta_1, \eta_2, \text{params}), \text{msk} = r$ 5. return (pp, msk) <p>KeyGen(pp, msk, f):</p> <ol style="list-style-type: none"> 1. parse $\text{pp} = (ek, \eta_1, \eta_2, \text{params}), \text{msk} = r$ 2. $\pi_f \leftarrow \text{zap.Priv}((\eta_1, \eta_2, f), r)$ 3. set $\text{sk}_f = (f, \pi_f)$ 4. return sk_f <p>Eval($\text{pp}, C, (\text{ct}_1, \dots, \text{ct}_\ell)$):</p> <ol style="list-style-type: none"> 1. parse $\text{pp} = (ek, \eta_1, \eta_2, \text{params})$ 2. parse $\text{ct}_i = (c_{v_i}, x_i, v_i, \text{pk}_{v_i}), \forall i \in [\ell]$ 3. for $i = 1$ to ℓ 4. $\widehat{c}_i \leftarrow \text{MFHE.Expand}((\text{pk}_{v_1}, \dots, \text{pk}_{v_\ell}), i, c_{v_i})$ 5. $\widehat{c} \leftarrow \text{MFHE.Eval}(\text{params}, C, (\widehat{c}_1, \dots, \widehat{c}_\ell))$ 6. return $\widehat{c} = (\widehat{c}, \{x_i, v_i, \text{pk}_{v_i}\}_{i \in [\ell]})$ 	<p>Enc(pp, x, m):</p> <ol style="list-style-type: none"> 1. parse $\text{pp} = (ek, \eta_1, \eta_2, \text{params})$ 2. $u \leftarrow \{0, 1\}^\lambda, v \leftarrow \text{PRG}(x \oplus u)$ 3. $y_v \leftarrow \text{WPRF.Eval}(ek, (x, v), u)$ 4. $(\text{pk}_v, \text{sk}_v) \leftarrow \text{MFHE.KeyGen}(\text{params}; y_v)$ 5. $c_v \leftarrow \text{MFHE.Enc}(\text{pk}_v, m)$ 6. return $\text{ct} = (c_v, x, v, \text{pk}_v)$ <p>Dec($\text{pp}, (\text{sk}_{f_1}, \dots, \text{sk}_{f_\ell}), \widehat{\text{ct}}$):</p> <ol style="list-style-type: none"> 1. parse $\text{pp} = (ek, \eta_1, \eta_2, \text{params})$ 2. parse $\text{sk}_{f_i} = (f_i, \pi_i), \forall i \in [\ell]$ 3. parse $\widehat{\text{ct}} = (\widehat{c}, \{x_i, v_i, \text{pk}_{v_i}\}_{i \in [\ell]})$ 4. for $i = 1$ to ℓ 5. $y_i \leftarrow \text{WPRF.Eval}(ek, (x_i, v_i), (\eta_1, \eta_2, f_i, \pi_i))$ 6. $(\text{pk}_i, \text{sk}_i) \leftarrow \text{MFHE.KeyGen}(\text{params}; y_i)$ 7. if $\text{pk}_i \neq \text{pk}_{v_i}$ 8. return \perp 9. return $\text{MFHE.Dec}(\text{params}, (\text{sk}_1, \dots, \text{sk}_\ell), \widehat{c})$
--	--

Fig. 13. Construction of multi-attribute pure FHE

5 Conclusion

We propose two generic approaches to construct IBE and ABE from WPRF, both of which are CCA secure and achieve a ciphertext of size $|m| + 2\lambda$. Existing schemes do not satisfy such optimal ciphertext size along with CCA security. Additionally, with the help of a pure MFHE, we convert our IBE and ABE into CCA1 secure MIFHE and MAFHE schemes respectively. Existing MIFHE and MAFHE [11] are CPA secure and rely on (possibly stronger assumption of) $i\mathcal{O}$.

References

1. Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudorandom bits. *SIAM J. Comput.* **13**(4), 850–864 (1984)
2. Boneh, D., Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput.* **36**(5), 1301–1328 (2007)
3. 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>
4. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) *PKC 2014*. LNCS, vol. 8383, pp. 501–519. Springer, Heidelberg (2014). <https://doi.org/10.1007/978-3-642-54631-0.29>
5. Brakerski, Z., Cash, D., Tsabary, R., Wee, H.: Targeted homomorphic attribute-based encryption. In: Hirt, M., Smith, A. (eds.) *TCC 2016*. LNCS, vol. 9986, pp. 330–360. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_13

6. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_29
7. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. *SIAM J. Comput.* **43**(2), 831–871 (2014)
8. Canetti, R., Raghuraman, S., Richelson, S., Vaikuntanathan, V.: Chosen-ciphertext secure fully homomorphic encryption. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10175, pp. 213–240. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54388-7_8
9. Chen, Y., Zhang, Z.: Publicly evaluable pseudorandom functions and their applications. *J. Comput. Secur.* **24**(2), 289–320 (2016)
10. Clear, M., Goldrick, C.M.: Attribute-based fully homomorphic encryption with a bounded number of inputs. *Int. J. Appl. Cryptography* **3**(4), 363–376 (2017)
11. Clear, M., McGoldrick, C.: Bootstrappable identity-based fully homomorphic encryption. In: Gritzalis, D., Kiayias, A., Askoxylakis, I. (eds.) CANS 2014. LNCS, vol. 8813, pp. 1–19. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12280-9_1
12. Clear, M., McGoldrick, C.: Multi-identity and multi-key leveled FHE from learning with errors. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 630–656. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_31
13. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 45–64. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_4
14. Daemen, J., Rijmen, V.: The Design of Rijndael: AES-The Advanced Encryption Standard. Springer, Heidelberg (2013). <https://doi.org/10.1007/978-3-662-04722-4>
15. Garg, S., Gentry, A., Sahai, C., Waters, B.: Witness encryption and its applications. In: Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, pp. 467–476. ACM (2013)
16. Gentry, C.: Practical identity-based encryption without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 445–464. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_27
17. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, pp. 169–178 (2009)
18. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5
19. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: How to run Turing machines on encrypted data. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 536–553. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_30
20. Goyal, R., Vusirikala, S., Waters, B.: Collusion resistant broadcast and trace from positional witness encryption. In: Lin, D., Sako, K. (eds.) PKC 2019. LNCS, vol. 11443, pp. 3–33. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17259-6_1

21. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 89–98 (2006)
22. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for NP. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 339–358. Springer, Heidelberg (2006). <https://doi.org/10.1007/11761679-21>
23. Halevi, S., Rogaway, P.: A tweakable enciphering mode. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 482–499. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_28
24. Halevi, S., Rogaway, P.: A parallelizable enciphering mode. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 292–304. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24660-2_23
25. Kiltz, E.: Direct chosen-ciphertext secure identity-based encryption in the standard model with short ciphertexts (2006)
26. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th Annual Symposium on Foundations of Computer Science (cat. No. 99CB37039), pp. 120–130. IEEE (1999)
27. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 735–763. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_26
28. Pal, T., Dutta, R.: Offline witness encryption from witness PRF and randomized encoding in CRS model. In: Jang-Jaccard, J., Guo, F. (eds.) ACISP 2019. LNCS, vol. 11547, pp. 78–96. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21548-4_5
29. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, pp. 387–394 (1990)
30. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, pp. 475–484. ACM (2014)
31. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_2
32. Yamada, S., Attrapadung, N., Hanaoka, G., Kunihiro, N.: Generic constructions for chosen-ciphertext secure attribute based encryption. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 71–89. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19379-8_5
33. Zhandry, M.: How to avoid obfuscation using witness PRFs. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 421–448. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0_16



Linear Complexity Private Set Intersection for Secure Two-Party Protocols

Ferhat Karakoç^{1(✉)} and Alptekin Küpçü²

¹ Ericsson Research, İstanbul, Turkey
ferhat.karakoc@ericsson.com

² Koç University, İstanbul, Turkey
akupcu@ku.edu.tr

Abstract. In this paper, we propose a new private set intersection (PSI) protocol with bi-oblivious data transfer that computes the following functionality. The two parties (P_1 and P_2) input two sets of items (X and Y , respectively) and one of the parties (P_2) outputs $f_i(b_i)$ for each $y_i \in Y$, where b_i is 0 or 1 depending on the truth value of $y_i \in X$ and f_i is defined by the other party (P_1) as taking 1-bit input and outputting the party's (P_1 's) data to be transferred. This functionality is generally required when the PSI protocol is used as a part of a larger secure two-party secure computation such as threshold PSI or any function of the whole intersecting set in general. Pinkas et al. presented a PSI protocol at Eurocrypt 2019 for this functionality, which has linear complexity only in communication. While there are PSI protocols with linear computation and communication complexities in the classical PSI setting where the intersection itself is revealed to one party, to the best of our knowledge, there is no PSI protocol, which outputs a function of the membership results and satisfies linear complexity in both communication and computation. We present the first PSI protocol that outputs only a function of the membership results with linear communication and computation complexities. While creating the protocol, as a side contribution, we provide a one-time batch oblivious programmable pseudo-random function based on garbled Bloom filters. We also implemented our protocol and provide performance results.

Keywords: Private set intersection · Two-party computation · Bloom filters · Oblivious transfer · Cuckoo hashing

1 Introduction

Private set intersection (PSI) protocols are one of the commonly used two party secure communication primitives where two parties, P_1 and P_2 , have their own respective private sets, X and Y , and at least one of the parties learn the intersection $X \cap Y$ but nothing more. In the last decade, considerable amount of

custom PSI protocols have been proposed in the literature. However, most of the proposed solutions reveal the intersection to at least one of the parties, which makes the protocols not usable as a building block in a larger secure computation protocol, because in that larger protocol, intermediate information would leak due to the nature of the employed PSI protocol. In this work, we focus on designing a PSI protocol in the semi-honest security model that outputs $\{f_i(b_i) \mid b_i = 1 \text{ if } y_i \in X, b_i = 0 \text{ otherwise}\}$ where f_i , defined by P_1 , takes 1-bit input and outputs P_1 's data to be transferred to P_2 . When each f_i is an identity function ($f_i(0) = 0$ and $f_i(1) = 1$), we obtain regular PSI. When f_i maps to a set of two strings, we obtain PSI with data transfer [7, 32]. While f_i appears to be general, it fails to cover general computation over $(X \cap Y)$, e.g., cardinality [6] or threshold PSI [32, 33], because each $f_i(b_i)$ is leaked individually to P_2 but not the computation over $(X \cap Y)$. Luckily, we can compose PSI with bi-oblivious data transfer with another layer of secure two-party computation protocol. For example, consider f_i as additively-homomorphic encryption of the identity function ($f_i(0) = E_k(0)$ and $f_i(1) = E_k(1)$ for key k picked by P_1), and that our protocol is followed by additively-homomorphic evaluation of the obtained values by P_2 , and then P_1 decrypts the result. This corresponds to PSI cardinality. Alternatively, f_i output values can be secret shares of the result for each item or labels for the corresponding input wires for circuit-based secure computation protocols. For example, the output of f_i can be a wire label for wire zero if $y_i \notin X$ and wire label for wire one if $y_i \in X$. This means that all wire labels are output for the larger protocol that employs our set intersection. This larger protocol can be, for example, computing a threshold over the intersection cardinality, or any other secure two-party computation protocol whose input should be the intersection.

The name *PSI with bi-oblivious data transfer* comes from the fact that f_i output values can be thought as the data to be transferred from P_1 to P_2 , but the transfer is bi-oblivious, meaning that neither P_1 nor P_2 knows the input bit b_i indicating which of the two data options was transferred.

Related Work: To the best of our knowledge, protocols that output a function of the membership results were proposed by Ciampi and Orlandi [5], Pinkas et al. [26], and Falk et al. [11] in addition to the circuit based solutions of [14, 29]. In [5], a custom private set membership protocol (PSM) (where one of the parties has only one item instead of a set) based on oblivious navigation of a graph was introduced and this PSM protocol was converted to a PSI protocol with $O(n \log n / \log \log n)$ communication and computation complexities using the hashing techniques proposed in [25, 28, 29], where n is the number of items in the sets. [11] has a communication complexity of $O(n \log \log n)$ when the output can be secret shared. In [26], Pinkas et al. proposed a PSI protocol with $O(n)$ communication and $\omega(n(\log \log n)^2)$ computation complexities using the oblivious programmable pseudo-random function (OPPRF) in [20]. That protocol uses OPPRF to check the private set membership relation in the hashed bins, where the result is not output in clear text, and then deploys a comparison circuit for the output of the membership result that can be given to a function as

the input. Also in literature, there have been special purpose PSI protocols such as [6, 8, 9, 15, 16, 19, 22, 31, 33], which output a specific function of the intersection such as cardinality of the set, intersection-sum, or a threshold function.

In our solution, we follow the idea of Pinkas et al. [26] in that we first run a PSM protocol for each bin in the cuckoo hash table and then execute a comparison protocol. We diverge from their idea in the following ways. The first one is that we construct a Bloom-filter (BF) based PSM protocol by modifying Dong et al. PSI solution [10] to reduce the computation complexity. The second point is that, instead of using a comparison circuit, we execute Ciampi-Orlandi PSM protocol as a secure equality testing protocol such as the one used in [18], which makes the equality testing free by using the base oblivious transfer already executed in the BF-based PSM protocol. Following these two methods along the idea of Pinkas et al., we are able to construct the first custom PSI protocol having linear computation and communication complexities in the number of items for the functionality we consider (outputting not the result set, but a function of the membership results), to the best of our knowledge. Note that there have been PSI solutions with linear complexities such as the protocols in [7, 10] and malicious secure solutions such as the recent proposals [13, 24] having linear communication complexity, but in these protocols the intersection is revealed to at least one party while in our protocol no party learns the intersection in clear-text. We implemented our PSM and PSI protocols and the Ciampi-Orlandi PSM protocol to make a fair comparison. Experimental performance results, which validate our performance analysis, are given in Sect. 7.

2 Preliminaries and Similar Protocols

Notation: P_1 and P_2 are the parties who run the protocol, X and Y are the corresponding item sets of the parties, and f is the function to be applied on the set intersection result. P_1 and P_2 respectively play the sender and receiver roles, and at the end of the PSI protocol, P_2 learns $f_i(b_i)$ for each $y_i \in Y$ where b_i takes a value from $\{0, 1\}$ depending on the truth value of the relation $y_i \stackrel{?}{\in} X$ and f_i is a function defined by P_1 with single-bit input.

The remaining notation we use throughout the paper is as follows:

- ℓ : The length of the items in the sets
- κ : Security parameter
- η : Statistical correctness parameter
- n : The number of items in the sets
- m : Bloom filter size
- k : Number of hash functions used in Bloom filter
- H_i : Set of k hash functions used in the construction of Bloom filters for i -th bin in the cuckoo table where $H_i = \{h_{i,1}, \dots, h_{i,k}\}$
- β : The number of bins in cuckoo table.

2.1 Sub-protocols

Oblivious Transfer: A 1-out-of-2 oblivious transfer (OT) [30] is a secure two-party protocol that realizes Functionality 1. While OT is one of the commonly used primitives in secure protocols, the main drawback of this primitive is the need of asymmetric key operation executions. With the help of OT extension (OTE) method proposed in [1] and practically realized with some studies such as [17], to execute 1-out-of-2 OT for m pairs of length ℓ (OT_ℓ^m) it is enough to run OT_κ^κ , called as base OTs, where κ is the security parameter, which keeps the number of heavy public key operations as a constant independent from the number of pairs m and item lengths ℓ .

In recent works, it was shown that the number of rounds can be 2 instead of 3 for an OT extension protocol by executing some of the computations in the offline phase of the protocol [3, 4]. In our solution, we don't consider the preprocessing operations and so we don't use these constructions in our protocols.

Cuckoo hashing: [23] is a hashing primitive that allows to map items of a set to the bins, where there is at most one item in each bin. This primitive employs two hash functions h_0 and h_1 and maps n items to a table T of $(1 + \epsilon)n$ bins. An item x_i is inserted into bin $T[h_b(x_i)]$. If this bin already accommodates a previous item x_j , then x_j is relocated to bin $T[h_{1-b}(x_j)]$. If in that bin there is another item, then this procedure is repeated until there is no need or a replacement threshold is reached. If a threshold is employed, then a stash is used to store the items that are not located into the bins.

Bloom Filter Based PSI: A Bloom filter (BF) [2] is a representation of a set $X = x_1, \dots, x_n$ of n elements using an m -bit string BF . BF is constructed with the help of a set of k independent and uniform hash functions ($H = h_1, \dots, h_k$) where $h_i : \{0, 1\}^\ell \rightarrow \{1, 2, \dots, m\}$ as follows: BF is first set to 0^m . Then, for each item in X , $BF[h_i(x_j)]$ is set to 1 where $1 \leq i \leq k$ and $1 \leq j \leq n$. To check whether an item x is in the set X , one checks $BF[h_i(x)]$ is equal to 1 or not for each i ($1 \leq i \leq k$). If for all i ($1 \leq i \leq k$) the corresponding bit in BF is equal to 1, then it means that the item is probably in the set. Otherwise (for some i the corresponding bit is 0), the item is not in the set.

A Bloom filter based PSI was proposed by Dong et al. [10]. In that solution, a variant of BF called as Garbled Bloom Filter (GBF) was used. A GBF of a set X , GBF , is similar to BF except that while for each hash function h_i in H we have $BF[h_i(x)] = 1$, $GBF[h_i(x)]$ is a secret share of x : that is, $\bigoplus_{i=1}^k GBF[h_i(x)] = x$ and other cells are random values instead of simple zeros. In the first step of the protocol, P_1 and P_2 construct a GBF (GBF_X) using the GBF building algorithm provided in [10] and a BF (BF_Y), respectively. Then, P_1 and P_2 run m -pair oblivious transfer of ℓ -bit strings (OT_ℓ^m) where P_1 's input is $(0^\ell, GBF_X[i])$ and P_2 's input is $BF_Y[i]$ for the i -th OT, and the output of P_2 is $GBF_Y[i]$. In this way, P_2 learns $GBF_X[i]$ if $BF_Y[i] = 1$. P_2 checks, for each item $y_j \in Y$, whether it is in X or not, by comparing $\bigoplus_{i=1}^k GBF_Y[h_i(y_j)] \stackrel{?}{=} y_j$.

Functionality 1. Oblivious Transfer

Inputs. The sender inputs a pair (x^0, x^1) , the receiver inputs a choice bit $b \in \{0, 1\}$

Outputs. The functionality returns the message x^b to the receiver and returns nothing to the sender

Oblivious Pseudo-random Function Based PSM: An oblivious pseudo-random function (OPRF), introduced in [12], is a two-party protocol where party P_1 holds a key K , party P_2 holds a string x , and at the end of the protocol P_1 learns nothing, while P_2 learns $F_K(x)$ where F is a pseudo-random function family that gets a κ -bit key K and an ℓ -bit input string x and outputs an ℓ -bit random-looking result. An oblivious programmable pseudo-random function (OPPRF) [20] is similar to an OPRF except that in OPPRF, the protocol outputs predefined values for some of the programmed inputs. In that protocol P_2 should not be able to distinguish which inputs are programmed. Note that OPPRF is very similar to PSI with data transfer [7, 32] by just setting the data of the latter to random values. Indeed, the GBF-based construction of OPPRF in [20] is essentially the GBF-based construction in [32]. In this paper, we extend this GBF-based construction to batch OPPRF.

The basic idea in OPRF based PSM protocols are as follows. P_1 holds a key K to compute a pseudo-random function F_K , P_2 learns $F_K(y)$ for his item y obliviously, and P_1 sends $F_K(x_i)$ for her items $x_i \in X$ to P_2 . P_2 checks if $F_K(y)$ is in the set $\{F_K(x_i)\}$. An example PSI protocol can be found in [29]. In the OPRF solution, P_2 learns whether or not his item is in the set of P_1 . This solution cannot be used in our setting where nobody learns the result in cleartext and the parties only learn a function result of the intersection. Pinkas et al. [26] converted the OPRF solution to the setting we consider using an oblivious programmable pseudo-random function. In that solution, P_1 sends the same (random) output r for the items in her set. Otherwise, she sends some random output to P_2 . Then P_1 and P_2 run a circuit to check the equality of r and the outputs P_1 sent to P_2 . At the end of this equality check circuit, one party obtains a function based on the result of the equality, i.e., of the membership.

Usage of Ciampi-Orlandi PSM Protocol to Test Equality of Two Strings: The private set membership (PSM) protocol proposed by Ciampi and Orlandi [5] works on the setting that P_1 and P_2 's inputs are a set of items X and an item y , respectively, and at the end of the protocol, P_2 learns a function of the membership relation and P_1 learns nothing. The protocol is based on oblivious graph tracing and uses oblivious transfer. In our construction, we use that protocol for the case that P_1 's input is just one item instead of a set, as considered in [18]. In this case, the PSM protocol becomes a secure equality testing outputting a function (we call functional equality testing - FEQT) protocol that realizes Functionality 2. This simplification also greatly increases efficiency, helping us achieve linear costs. Protocol 1 presents the steps of Ciampi-Orlandi PSM protocol for the case of testing two strings as used in [18].

2.2 Security Definitions

Since there are two parties who run the protocol, it is enough to prove that the protocol is secure when one of the parties is corrupted. There are two possible cases: either P_1 or P_2 is corrupted.

Functionality 2. Functional Secure Equality Testing

Inputs. P_1 inputs x and a function f to be computed on the equality relation result, P_2 inputs y

Outputs. The functionality checks the equality of x and y and returns $f(0)$ or $f(1)$ according to the truth value of $x \stackrel{?}{=} y$ to P_2

Protocol 1. (Ciampi-Orlandi PSM Protocol to test equality of two strings.)

Parameters. $E_k(\cdot)$ is a symmetric encryption under the key k with a polynomial-time verification algorithm outputting whether a given ciphertext is in the range of $E_k(\cdot)$ with false positive probability being $2^{-\eta}$.

Inputs. P_1 inputs x and a function f to be computed on the equality relation result, P_2 inputs y .

Outputs. P_2 outputs $f(0)$ or $f(1)$ according to the truth value of $x \stackrel{?}{=} y$. P_1 outputs nothing.

The protocol steps:

1. P_1 prepares the message pairs (S_0^i, S_1^i) for $x[i]$ ($1 < i < \ell$) as follows: ($x[i]$ denotes the i -th bit of x and $x[1]$ is the right-most bit)
 - chooses random symmetric keys k_ℓ and k_ℓ^* and sets $S_{x[\ell]}^\ell = k_\ell$ and $S_{1-x[\ell]}^\ell = k_\ell^*$
 - For $i = (\ell - 1)$ to 1
 - chooses random symmetric keys k_i and k_i^* and sets $S_{x[i]}^i = \{E_{k_{i+1}}(k_i), E_{k_{i+1}^*}(k_i^*)\}$ and $S_{1-x[i]}^i = \{E_{k_{i+1}}(k_i^*), E_{k_{i+1}^*}(k_i)\}$.
 - permutes the ciphertexts in $S_{x[i]}^i$ and $S_{1-x[i]}^i$ randomly.
2. P_1 sends $E_{k_1}(f(1))$ and $E_{k_1^*}(f(0))$ to P_2 in random order.
3. P_2 learns corresponding $S_{y[i]}^i$'s by running OT from P_1 for $1 < i < \ell$.
4. P_2 recovers only one of the keys k_1 or k_1^* by decrypting the ciphertexts in the following way:
 - decrypts the ciphertexts in $S_{y[\ell-1]}^{\ell-1}$ using $S_{y[\ell]}^\ell$ as the key where the plaintext in the encryption domain is the key that will be used to decrypt the ciphertexts in $S_{y[\ell-2]}^{\ell-2}$.
 - decrypts the ciphertexts in $S_{y[i]}^i$ using the plaintext recovered from $S_{y[i+1]}^{i+1}$ as the key to recover the key used in the next received message $S_{y[i-1]}^{i-1}$.
5. P_2 decrypts the ciphertexts $E_{k_1}(f(1))$ and $E_{k_1^*}(f(0))$ using the key recovered in Step 4 where only one of the plaintexts will be in the domain and this plaintext will be equal to $f(1)$ or $f(0)$. P_2 outputs the result.

We follow the simulation-based security proof paradigm. Since we only consider honest-but-curious adversaries, the existence of a simulation in the “ideal world” whose protocol transcript is computationally indistinguishable from the adversary’s view in the protocol execution in the “real world” (together with the parties’ outputs in both worlds) proves that the protocol is secure. The basic idea in this proof paradigm is that if it is possible for the simulator to create a protocol transcript indistinguishable from the real execution transcript, then the transcript doesn’t reveal any piece of information about the private input of the honest party. This security proof paradigm was formalized in [21] as follows. Protocol π implements the functionality $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ where the output of P_1 and P_2 are $\mathcal{F}_1(x, y)$ and $\mathcal{F}_2(x, y)$, respectively, and x and y are the inputs of the parties. The view of P_i for $i \in \{1, 2\}$ (denoted as $view_i^\pi(x, y)$) in the execution of the protocol π is the input of P_i , the internal random number coin tosses, the messages received from the other party in the execution of the protocol, and the outputs. The existence of probabilistic polynomial-time (PPT) algorithms S_i (the simulators) that takes the input of P_i and the output of P_i such that

$$\{S_i(w_i, \mathcal{F}_i(x, y))\}_{x,y} \approx \{view_i^\pi(x, y)\}_{x,y}$$

for $i \in \{1, 2\}$ where $w_1 = x$ and $w_2 = y$ proves that the protocol π realizes the functionality \mathcal{F} securely.

As for the underlying primitives, namely OT and FEQT, whose functionalities were presented as Functionalities 1 and 2, respectively, there exists simulators who can simulate the view for both parties. These simulators take the input and output of the corresponding party as input, and produce indistinguishable views as output. In our proofs, we make use of these simulators for the underlying primitives.

Lastly, in our proofs, we provide the simulators for semi-honest adversaries. Note that the simulated view (including the outputs) must be indistinguishable from the real view. In all our proofs, this is either obvious (directly comes from the security of the underlying primitive, or comes from the fact that the simulated values are picked from the same distribution as the original ones), or were proven by others (in which case we also cite those papers). Thus, we do not delve deep into the indistinguishability discussions, considering also the page limits.

3 Bloom Filter Based OPPRF Construction

We present a one-time OPPRF construction based on PSI protocols proposed in [10] and [32]. For our usage, we put secret shares of random values chosen by the sender as the data to be transferred by the PSI protocol [32].

The OPPRF functionality we use in our PSM protocol is given in Functionality 3 and our construction that implements the functionality is presented in Protocol 2. The probability of false negative is zero because when $y \in X$, P_2 learns all shares required to recover the related programmed value. There may be false positives only with probability that is negligible in k and η , where k is the number of hash functions used in GBF construction and η is the minimum

bit length of each cell in GBF, as shown in [10]. Note that we allow the programmed values (t_i) to be correlated. Because of that, the functionality is secure only if the receiver makes only one query. For the purposes of PSM, we notice that one query is enough. In our PSM solution the programmed values will be the same; that is, all the t_i values will be equal.

Asymptotic Complexity. Since the number of hash functions used in the construction of Bloom filters is a constant related to the statistical correctness parameter that is independent of the number of items, Protocol 2 requires $O(n)$ hash function computations for the construction of garbled Bloom filter in Step 1. Also, the size of the Bloom filters is $m = O(n)$, which makes the total asymptotic complexity of running oblivious transfers in Step 3 $O(n)$. Step 2 requires $O(n)$ non-cryptographic computation and space. Considering the complexity of Step 4 as $O(1)$, we conclude that the OPPRF protocol has a communication, computation, and space complexity of $O(n)$.

Functionality 3. (One-Time) Oblivious Programmable Pseudo Random Function

Inputs. P_1 inputs predefined items $X = \{x_1, \dots, x_n\}$ and corresponding programmed values $T = \{t_1, \dots, t_n\}$, P_2 inputs y

Outputs. The functionality checks the membership $y \in X$ and returns t_i to P_2 if $\exists x_i$ s.t. $y = x_i$ ($1 \leq i \leq n$); returns a random value otherwise to P_2 , and returns nothing to P_1

Protocol 2. Our One-Time OPPRF Protocol

Parameters. A set of hash functions $H = \{h_1, \dots, h_k\}$

Inputs. P_1 inputs a set of items $X = \{x_1, \dots, x_n\}$ and corresponding programmed values $T = \{t_1, \dots, t_n\}$, P_2 inputs an item y .

Outputs. P_1 outputs nothing and P_2 outputs t_i if $\exists x_i$ s.t. $y = x_i$ ($1 \leq i \leq n$), otherwise outputs a random value.

The protocol steps:

1. P_1 constructs a garbled Bloom filter GBF_X having $\max(\eta, \ell)$ -bit strings in each cell such that

$$\bigoplus_{i=1}^k GBF_X[h_i(x_j)] = t_j$$

for $1 \leq j \leq n$.

2. P_2 constructs a (standard) Bloom filter BF_y for the item y .
 3. P_1 and P_2 run m oblivious transfers where P_1 's input is $(0, GBF_X[i])$ and P_2 's input is $BF_y[i]$ for the i -th oblivious transfer, and the output of P_2 is 0 if $BF_y[i] = 0$ or $GBF_X[i]$ if $BF_y[i] = 1$. Call the output of P_2 as $GBF_y[i]$.
 4. P_1 outputs nothing and P_2 outputs $\bigoplus_{i=1}^k GBF_y[h_i(y)]$.
-

Theorem 1. *Protocol 2 securely realizes Functionality 3 when P_1 is corrupted by a semi-honest adversary \mathcal{A} , assuming that the OT protocol is semi-honest secure.*

Proof. The input set X and the programmed values T are given to the simulator S . The simulator computes a garbled Bloom filter GBF_X using its random tape such that $\bigoplus_i^k GBF_X[h_i(x_j)] = t_j$ for $1 \leq j \leq n$. S runs the simulator of OT as the sender m times, where for the i -th run, the input of the simulator is $((0, GBF_X[i]), \perp)$. Here, $(0, GBF_X[i])$ is the input of the sender in the OT protocol and there is no output of the sender. Thus, the simulated view and output of the parties, and the view of the adversary in the real execution of the protocol and the output of the parties are indistinguishable.

Theorem 2. *Protocol 2 securely realizes Functionality 3 when P_2 is corrupted by a semi-honest adversary \mathcal{A} , assuming that the OT protocol is semi-honest secure.*

Proof. The input item y and the output $\bigoplus_{i=1}^k GBF_y[h_i(y)]$ are given to the simulator S . The simulator constructs the Bloom filter using y regularly, and creates GBF'_y by running the following steps:

1. Set random values to $GBF'_y[h_i(y)]$ for $1 \leq i < k$.
2. Set $GBF'_y[h_k(y)] = \bigoplus_{i=1}^k GBF_y[h_i(y)] \oplus \bigoplus_{i=1}^{k-1} GBF'_y[h_i(y)]$.
3. Set $GBF'_y[i] = 0$ if $BF_y[i] = 0$.

Finally, S runs the OT simulator as the receiver m times, where in the i -th, run the receiver's input is $BF_y[i]$ and the receiver's output is $GBF'_y[i]$. The proof concludes when we show that GBF'_y is indistinguishable from GBF_y . The cells in both GBF'_y and GBF_y are equal to '0' for the indices i where $BF_y[i] = 0$. Now we need to show that for the remaining k cells these GBFs are indistinguishable. Any combination of $(k-1)$ cells are random due to the property of secret sharing and the xor of k cells equals to $\bigoplus_{i=1}^k GBF_y[h_i(y)]$ in both GBFs, which concludes the proof.

4 Our Private Set Membership Protocol

In this section, we propose a new PSM protocol that realizes Functionality 4. As discussed in the introduction, our protocol does not output the membership result, but instead outputs some function of it, so that it can be directly integrated into a larger secure computation protocol. After this section, we show how to extend our protocol to set intersection as well.

In the construction of the protocol, we use the following idea of [26]: If $y \in X$, then both parties learn the same random value. Otherwise, they learn different random values. Then, the parties run a comparison protocol that outputs a function of the equality instead of the equality itself (Functionality 2). Our solution diverges from the solution of [26] in two folds. To realize the first part, [26] makes

use of an OPPRF construction based on polynomials. We propose a new OPPRF construction based on Bloom filters. The selection of Bloom filters enables us to reduce the computation complexity of the protocol to a linear complexity. The other difference is that we utilize Ciampi-Orlandi PSM protocol [5] for secure equality testing as done in [18] for Functionality 2, instead of running a comparison circuit. Thus, our overall construction is not a circuit-based construction.

Functionality 4. Private Set Membership

Inputs. P_1 inputs $X = \{x_1, \dots, x_n\}$ and a function f to be computed on the membership relation result, P_2 inputs y .

Outputs. The functionality checks the membership of y in X and returns $f(1)$ to P_2 if $y \in X$. Otherwise, returns $f(0)$ to P_2

The overall view of our PSM protocol is as follow. To achieve private set membership, the parties first run the one-time OPPRF protocol based on garbled Bloom filters, where P_1 outputs r (a random value chosen by P_1), whereas P_2 learns some random value that may be r or something different. The value P_2 learns is always random and indistinguishable; but, this random value is equal to r if and only if $y \in X$. Following this part, the parties run a secure functional equality testing protocol, where at the end of the protocol P_2 learns the function result of the equality relation, which is also the function result of the membership relation. We make use of the PSM protocol of Ciampi-Orlandi [5] for secure functional equality testing by reducing the number of items of the sender set to one. We present our semi-honest secure PSM solution in Protocol 3.

Protocol 3. Our Private Set Membership Protocol

Parameters. A set of hash functions $H = \{h_1, \dots, h_k\}$.

Inputs. P_1 inputs a set of items $X = \{x_1, \dots, x_n\}$ and a function f to be computed on the membership relation result, P_2 inputs an item y .

Outputs. P_2 outputs $f(1)$ if $y \in X$. Otherwise, P_2 outputs $f(0)$. P_1 outputs nothing.

The protocol steps:

1. P_1 picks an η -bit random value r and sets $T = \{t_1 = r, \dots, t_n = r\}$.
 2. P_1 and P_2 run Protocol 2 for one-time OPPRF with the respective inputs (X, T) and y . Denote the output of P_2 as r' .
 3. P_1 and P_2 run Protocol 1 for functional equality testing with the respective inputs r and r' . The output of the PSM protocol is the output of Protocol 1.
-

Asymptotic Complexity of Our PSM Protocol. Protocol 2 requires $O(n)$ hash function computations as stated in the previous section. FEQT employs $O(\eta)$ operations for η oblivious transfers in the Ciampi-Orlandi PSM protocol. Thus,

the asymptotic computation complexity of our PSM protocol becomes $O(n)$. The communication complexity comes from the oblivious transfers. Considering the oblivious transfer extension communication complexity as linear in the number of OTs, the communication complexity of Protocol 3 is also $O(n)$.

Theorem 3. *Protocol 3 securely realizes Functionality 4 when P_1 is corrupted by a semi-honest adversary \mathcal{A} , assuming that the OPPRF and FEQT protocols are semi-honest secure.*

Proof. The simulator S is given the input set X . S picks a random value r using its random tape and sets $T = \{t_1 = r, \dots, t_n = r\}$. The simulator S runs the simulator of OPPRF protocol with the input $((X, T), \perp)$. Then, S runs the simulator of FEQT protocol with the input (r, \perp) . This completes the whole simulation, and indistinguishability is a direct result of the underlying simulators.

Theorem 4. *Protocol 3 securely realizes Functionality 4 when P_2 is corrupted by a semi-honest adversary \mathcal{A} , assuming that the OPPRF and FEQT protocols are semi-honest secure.*

Proof. The simulator S is given the input item y and the output $f(b)$ for $b = y \stackrel{?}{\in} X$. The simulator picks a η -bit random value r'' . S runs the simulator of OPPRF with the input (y, r'') and the simulator of FEQT with the input $(r'', f(b))$. S does not know the uniform random value r' used in the real execution, but it follows the same distribution as r'' , and therefore they are perfectly indistinguishable. The computational indistinguishability comes from the FEQT and OPPRF simulations, which are based on OT simulations.

5 Batch One-Time OPPRF

We propose a new batch one-time OPPRF construction in Protocol 4 that implements Functionality 5, to be used in our PSI protocol. For the construction of a batch OPPRF from Protocol 2, instead of using different garbled Bloom filters for each programmed value set, we construct only one garbled Bloom filter, and store the shares of programmed values in the same garbled Bloom filter. Note that for each set X_i , a different set of hash functions (hash function set H_i for the programmed value set X_i) is used, since there might be some items which belong to more than one set.

Asymptotic Complexity. Since the size of the garbled Bloom filter is linear in the number of items to be stored in it and OT extension is also linear in the number of OT executions, the computation and communication complexities of our batch one-time OPPRF protocol becomes linear in the total number of programmed values in the programmed value sets.

Functionality 5. Batch One-Time Oblivious Programmable Pseudo Random Function

Inputs. P_1 inputs a predefined set of item sets $X = \{X_1, \dots, X_\beta\}$, where $X_i = \{x_{i,1}, \dots, x_{i,n}\}$, and corresponding programmed value sets $T = \{T_1, \dots, T_\beta\}$, where $T_i = \{t_{i,1}, \dots, t_{i,n}\}$, and P_2 inputs a set of items $Y = \{y_1, \dots, y_\beta\}$

Outputs. The functionality checks the membership relations $y_i \in X_i$ and returns $r'_i = t_{i,j}$ if $\exists x_{i,j}$ s.t. $y_i = x_{i,j}$ ($1 \leq j \leq n$); returns a random r'_i otherwise, for each i where $1 \leq i \leq \beta$

Protocol 4. Bloom Filter Based Batch One-Time OPPRF Protocol

Parameters. A set of hash function sets $H = \{H_1, \dots, H_\beta\}$ where

$$H_i = \{h_{i,0}, \dots, h_{i,k}\}$$

Inputs. P_1 inputs a set of item sets $X = \{X_1, \dots, X_\beta\}$, where

$X_i = \{x_{i,1}, \dots, x_{i,n}\}$, and corresponding programmed value sets

$T = \{T_1, \dots, T_\beta\}$, where $T_i = \{t_{i,1}, \dots, t_{i,n}\}$, and P_2 inputs a set of items

$Y = \{y_1, \dots, y_\beta\}$.

Outputs. P_2 outputs a set of random values $R' = \{r'_1, \dots, r'_\beta\}$, where $r'_i = t_{i,j}$ if $\exists x_{i,j}$ s.t. $y_i = x_{i,j}$ ($1 \leq j \leq n$); otherwise r'_i is a random value; for $1 \leq i \leq \beta$.

The protocol steps:

1. P_1 constructs a garbled Bloom filter GBF_X having $\max(\eta, \ell)$ -bit strings in each cell such that

$$\bigoplus_{j=1}^k GBF_X[h_{i,j}(x_{i,l})] = t_{i,l}$$

for $1 \leq i \leq \beta$ and $1 \leq j \leq k$.

2. P_2 constructs a Bloom filter BF_Y for the items in Y .
 3. P_1 and P_2 run m oblivious transfers where P_1 's input is $(0, GBF_X[i])$ and P_2 's input is $BF_Y[i]$ for the i -th oblivious transfer, and the output of P_2 is 0 if $BF_Y[i] = 0$ or $GBF_X[i]$ if $BF_Y[i] = 1$. Call the OT output P_2 obtains as $GBF_Y[i]$.
 4. P_2 outputs $R' = \{r'_1, \dots, r'_\beta\}$ where $r'_i = \bigoplus_{j=1}^k GBF_Y[h_{i,j}(y_i)]$.
-

Theorem 5. Protocol 4 securely realizes Functionality 5 when P_1 is corrupted by a semi-honest adversary \mathcal{A} , assuming that the OT protocol is semi-honest secure.

Proof. The simulator S is given the input set of sets X and the programmed values set T . The simulator computes a garbled Bloom filter using its random tape such that $\bigoplus_{j=1}^k GBF_X[h_{i,j}(x_{i,l})] = t_{i,l}$. S runs the simulator of the OT protocol as the sender with the input (GBF_X, \perp) . This concludes the simulation. Indistinguishability directly comes from the garbled Bloom filter construction following the protocol, and the OT simulator.

Theorem 6. *Protocol 4 securely realizes Functionality 5 when P_2 is corrupted by a semi-honest adversary \mathcal{A} , assuming that the OT protocol is semi-honest secure.*

Proof. The input set Y and the output R' are given to the simulator S . The simulator constructs a Bloom filter for Y and a garbled Bloom filter GBF'_Y following the steps:

1. Constructs a BF BF_Y for Y .
2. Constructs a GBF GBF'_Y such that $\bigoplus_{j=1}^k GBF'_Y[h_{i,j}(y_i)] = r'_i$ for $1 \leq i \leq \beta$.
3. Sets $GBF'_Y[i] = 0$ if $BF_Y[i] = 0$.

Then, S runs the simulator of the OT protocol as the receiver with the input (BF_Y, GBF'_Y) . Note that the garbled bloom filters GBF'_Y and GBF_Y are indistinguishable as discussed in the proof of Theorem 2.

6 Our Private Set Intersection Protocol

Our PSM protocol can be used to build an efficient PSI protocol using the hashing techniques introduced in [25, 28]. In this technique, one party constructs a cuckoo table as mentioned in Sect. 2.1 using two hash functions and the other party maps her items into bins in a hash table using the two hash functions that are applied on each item. Then, a private set membership protocol is applied on each bin where the party who constructs the cuckoo table inputs the (single) item in the i -th bin, and the other party inputs the set of items in the i -th bin of its hash table, for the i -th execution of the PSM protocol. If one were to directly employ our PSM construction to obtain a PSI protocol using this hashing technique, the computation and communication complexities of the full PSI protocol would be $O(n \log n / \log \log n)$, since the number of items in each hash table bin is $O(\log n / \log \log n)$ and the number of bins is $O(n)$. Note that with this usage, for each bin, P_2 and P_1 run $O(n)$ parallel OPPRF protocols and then apply $O(n)$ parallel FEQT protocols. Instead of following this straightforward way, we show that it is possible to make the communication and computation complexities linear while extending our PSM solution to a PSI solution using our batch one-time OPPRF protocol.

Our full PSI protocol that realizes Functionality 6 is introduced in Protocol 5. Note that in Step 4 of the protocol, the items in the bins of the hash table are given as the programmed values in the programmed value sets to the batch one-time OPPRF protocol. While there are many items in the bins of the hash table, most of them are random values and the total number of non-random items in the hash table will be the product of the number of items (n) and the number of cuckoo hash functions (chosen as 3 in our protocol). Thus, the size of the garbled Bloom filter constructed in the batch one-time OPPRF protocol will be $O(n)$, which allows our PSI protocol to have linear complexity.

Functionality 6. Private Set Intersection

Inputs. P_1 inputs $X = \{x_1, \dots, x_n\}$ and a set of functions $\{f_1, \dots, f_n\}$, P_2 inputs $Y = \{y_1, \dots, y_n\}$

Outputs. Returns $\{f_i(b_i) \mid 1 \leq i \leq n\}$ to P_2 where $b_i = 1$ if $y_i \in X$, $b_i = 0$ otherwise

Protocol 5. Bloom Filter Based Private Set Intersection Protocol

Parameters. A set of hash function sets $H = \{H_1, \dots, H_\beta\}$ where $H_i = \{h_{i,1}, \dots, h_{i,k}\}$ for $1 \leq i \leq \beta$.

Inputs. P_1 inputs a set of items $X = \{x_1, \dots, x_n\}$ and a set of functions $\{f_1, \dots, f_n\}$, P_2 inputs a set of items $Y = \{y_1, \dots, y_n\}$.

Outputs. P_2 outputs $\{f_i(b_i) \mid 1 \leq i \leq n\}$ where $b_i = 1$ if $y_i \in X$, $b_i = 0$ otherwise.

The protocol steps:

1. P_1 constructs a hash table for the set X .
2. P_2 constructs a cuckoo table for the set Y .
3. P_1 picks a set of β η -bit random values $R = \{r_1, \dots, r_\beta\}$.
4. P_1 and P_2 run Protocol 4 with their respective inputs: (hash table, R) and cuckoo table. Let the output of P_2 be $R' = \{r'_1, \dots, r'_\beta\}$.
5. P_1 and P_2 run β parallel executions of Protocol 1 for functional equality testing, where for the i -th run, the inputs of P_1 and P_2 are r_i and r'_i . For each item y_j in Y , P_2 outputs the i -th execution output of Protocol 1 where y_j is placed into i -th bin in the cuckoo table.

Note that when we use two hash functions for cuckoo hashing, then there will be some items in Y which cannot be placed into the table and have to be moved to a stash. For each of these items in the stash, a PSM protocol also has to be executed. When we consider the number of these items as $\omega(1)$, then the complexity of our PSI protocol becomes bigger than $O(n)$. To make the complexity linear, Pinkas et al. proposed to use dual execution or a stash-less cuckoo hashing [26]. In dual execution, after the first run of the PSI protocol, P_2 learns the membership result for its items except the ones in the stash. Then the parties run the PSI protocol swapping their roles, that is, P_1 constructs a cuckoo table for X and P_2 constructs a hash table for the items in the stash. Since there may be some items of P_1 which have not been placed in the cuckoo table and moved to a stash, P_1 and P_2 should run the PSM protocol for their items in the stashes. However, this usage does not realize the Functionality 6 that we consider, since in the second run, P_2 learns the function of the membership result between its items in the stash and the set X , and in the final PSM protocols run for the items in the stashes, P_2 again learns the function of the membership result between its items in the stash and P_1 's items in the stash. That is, P_2 learns two different results for its items in the stash that makes the protocol diverge from Functionality 6. Because of these two reasons, we make use of the second

method of Pinkas et al., which is the usage of stash-less cuckoo hashing with three hash functions.

Asymptotic Complexity. While it seems that there are $O(n \log n / \log \log n)$ items in the hash table of P_1 , which makes the length of the Bloom filters $O(n \log n / \log \log n)$, the actual number of items is $O(3n) = O(n)$ since the other items are random values padded to the bins to make the number of items in the bins $O(\log n / \log \log n)$. Thus, the complexity of Step 4 of Protocol 5 becomes $O(n)$. Since the number of bins is $O(n)$ and for each bin only one equality testing is executed in Step 5, the complexity of Step 5 will be $O(n)$. Thus the communication and computation complexities of our PSI protocol becomes $O(n)$.

Theorem 7. *Protocol 5 securely realizes Functionality 6 when P_1 is corrupted by a semi-honest adversary \mathcal{A} , assuming that the batch OPPRF and FEQT protocols are semi-honest secure.*

Proof. The input set X is given to the simulator S . The simulator computes the hash table for X and picks β η -bit random values $R'' = \{r''_1, \dots, r''_\beta\}$ using its random tape. Then S runs the simulator of batch OPPRF protocol with the input $((\text{hash table}, R), \perp)$. Finally, S runs the simulator of FEQT protocol β times, where the input in the i -th run is (r_i, \perp) . Since r''_i and r_i are random numbers from a uniform distribution, they are indistinguishable. Hence, indistinguishability of S follows the indistinguishability of the underlying batch OPPRF and FEQT simulators.

Theorem 8. *Protocol 5 securely realizes Functionality 6 when P_2 is corrupted by a semi-honest adversary \mathcal{A} , assuming that the batch OPPRF and FEQT protocols are semi-honest secure.*

Proof. The simulator S is given the input set Y and the output $f_i(b_i)$ for $1 \leq i \leq n$. S computes a cuckoo table for the set Y and picks β η -bit random values $R'' = \{r''_1, \dots, r''_\beta\}$. S runs the simulator of batch OPPRF protocol with the input $(\text{cuckoo table}, R'')$ and the simulator of FEQT protocol β times, where the input in the i -th run is $(r''_i, f_j(b_j))$ where y_j is assigned to the i -th bin. Since R' in the real execution and R'' in the ideal world are uniformly selected sets of random numbers, they are indistinguishable. Hence, indistinguishability of S follows the indistinguishability of the underlying batch OPPRF and FEQT simulators.

7 Performance Evaluation

7.1 Concrete Complexity

Parameter Choices. We take the number of hash functions used in the construction of Bloom filters as $k = \eta$ and follow the choice of [10] to set the size of the Bloom filter as taking $m = 1.44kn$. Note that taking $k = \eta$ doesn't reduce the security level to statistical correctness parameter because the result of BF-based

OPPRF protocol are random numbers which then be inputs of the equality testing protocol. Following the parameters in [26], we choose the number of bins as $1.27n$ and the number of cuckoo hashes as 3, which makes the probability of having at least one item in the stash 2^{-40} , consistent with our preferred statistical correctness parameter η .

Concrete Complexity of Our PSM Protocol. For the Bloom filters, P_1 and P_2 compute nk and k hash functions, respectively. For the OT-extension in the OPPRF part, they run m oblivious transfer whose total computation complexity is approximately equal to $3m$ symmetric key operations thanks to the oblivious transfer extension [17]. Finally, the parties execute Ciampi-Orlandi PSM protocol where the number of items in the set of P_1 is one, which makes the computation complexity 6η symmetric key operations at P_1 and 5η symmetric key operations at P_2 (the reader can refer to [18] for the complexity calculation for the FEQT protocol)¹. Thus the computation complexity of the protocol at the party where majority of workload is done is $nk + 3m + 6\eta$. Since we choose $m = 1.44kn$ and $k = \eta$ then the complexity becomes $5.32n\eta + 6\eta$. For the parameter $\eta = 40$ the complexity will be $212.8n + 240$ symmetric key operations. The communication complexity comes from the oblivious transfers. In the OPPRF step, the message lengths in the oblivious transfer is η bits, while for the FEQT part, it is $2(\kappa + \eta)$ bits. Considering that the total number of bits transferred in the OT extension equals to 2 times the items' length times number of pairs, the communication complexity of the protocol becomes $2 \times m \times \eta + 2 \times \eta \times 2 \times (\kappa + \eta) = 2 \times (1.44 \times n \times \eta) \times \eta + 2\eta \times 2 \times (\kappa + \eta) = 2.88n\eta^2 + 4\kappa\eta + 4\eta^2$.

Concrete Complexity of Our PSI Protocol. To construct the cuckoo hash table, P_2 and P_1 perform at most $3n$ hash operations. Then, they construct BF performing kn and $3kn$ hash computations, respectively. They execute $4.32n\eta$ OTs using OT extension, which costs $3 \times 4.32n\eta$ hash computations. In the last step, P_1 and P_2 perform $1.27n \times (5\eta)$ and $1.27n \times (6\eta)$ hash computations, respectively. Thus the total computation cost on the party who has the maximum overhead is $3n + 3nk + 3 \times 4.32n\eta + 1.27n \times (6\eta) = 26.58n\eta + 3n$. The communication cost comes from the OT executions for Bloom filter and equality test. Since for the Bloom filter, $4.32n\eta$ η -bit message pairs are obviously sent, the dominant cost is $2 \times 4.32n\eta \times \eta = 8.64n\eta^2$. For the equality test, the length of the pairs is $2 \times (\kappa + \eta)$ and the number of pairs is $1.27n$; hence, the dominant part is $2 \times 1.27n \times 2(\kappa + \eta)$. Thus, the total communication cost is approximately is $8.64n\eta^2 + 5.08n(\kappa + \eta)$.

7.2 Experimental Verification

Setup. We implemented Ciampi-Orlandi and our PSM protocol using C programming language and GMP library. In our experiment setup, P_1 and P_2 run

¹ The item lengths in the GBF and so the lengths of the items to be tested for equality are $\max(\eta, \ell)$ bits as stated in Protocol 2. In concrete complexity analysis, we take it as η for simplicity considering that in practice generally $\eta > \ell$.

on the same machine as different processes and communicate with each other over a TCP channel. We run the protocols for different size of sets and item lengths on a single CPU core of a computer that has 2.1 GHz 16-core Intel Xeon CPU with 64 GB RAM. In the experiments, we chose RSA 2048 as asymmetric encryption algorithm in base OT, the statistical correctness parameter η as 40 bits, AES as the encryption algorithm, SHA-256 with different initialization vectors as the hash functions. We take the f function such that it outputs 128-bit wire labels. We take the number of hash functions in the construction of Bloom filters in our protocols as $k = 40$. The results are the averages over 10 executions of the protocols.

PSM. Table 1 shows the total amount of data transmitted between P_1 and P_2 during the execution of the protocols and the run-times in LAN and WAN setting. As can be seen from the table, our BF-based semi-honest PSM protocol has linear complexity both on computation and communication, and we provide comparable performance. Our asymptotic advantage becomes visible with larger ℓ values.

Table 1. Performance results of Ciampi-Orlandi and our PSM protocols. Run-time estimates are done for LAN and WAN under the assumption that the bandwidth in LAN (respectively in WAN) is 1 Gbps (100 Mbps) and RTT is 1 ms (100 ms).

Protocol		Ciampi-Orlandi PSM			Our PSM		
Set size n		$n = 2^{12}$	$n = 2^{14}$	$n = 2^{16}$	$n = 2^{12}$	$n = 2^{14}$	$n = 2^{16}$
Comm. [MB]	$\ell = 32$	5.4	21.3	84.5	6.0	23.6	93.8
	$\ell = 48$	8.1	31.8	126.5	6.5	25.4	101.0
	$\ell = 64$	10.7	42.3	168.5	7.4	29.0	115.4
LAN [ms]	$\ell = 32$	2045	4195	11717	2583	6508	22031
	$\ell = 48$	2444	5759	18115	2655	6564	22337
	$\ell = 64$	2793	7361	24396	2656	6599	22542
WAN [ms]	$\ell = 32$	10207	36389	139436	11651	42118	163745
	$\ell = 48$	14686	53824	209315	12419	44895	174973
	$\ell = 64$	18966	71296	279050	13781	50371	196904

PSI. We also implemented our PSI protocol to validate our performance analysis and compare the efficiency of our protocol with the existing solutions. We choose the number of hash functions in cuckoo hashing as 3, the number of bins as $1.27n$, and the size of the BF as $n \times 1.44 \times 3 \times k$ where k is the number of hash functions used in BF and 3 comes from the number of hash functions in cuckoo hashing. We evaluated the effect of k on the performance of our PSI protocol running the protocol for different k values which is related to the correctness of our protocol. The results are given in Table 2.

Table 2. Effect of number of hash functions in Bloom filter on the performance of our PSI protocol.

	Comm [MB]			LAN [ms]			WAN [ms]		
	$n = 2^{10}$	$n = 2^{12}$	$n = 2^{14}$	$n = 2^{10}$	$n = 2^{12}$	$n = 2^{14}$	$n = 2^{10}$	$n = 2^{12}$	$n = 2^{14}$
$k = 40$	9.4	36.9	147.4	2604	6804	22488	16812	62577	245277
$k = 60$	11.5	45.7	182.5	3018	8586	28725	20400	77660	304566
$k = 80$	13.8	54.5	217.6	3545	10278	35086	24403	92652	363980

We run our PSI protocol for different item bit lengths and set sizes choosing $k = 40$, which satisfies enough correctness in practical applications, and obtained the results in Table 3. The table verifies our complexity claims and shows that our PSI protocol has linear communication and computation complexities. We also present the linear trend in computation complexity of our protocol in Fig. 1 where the numbers are taken from Table 3 for $\ell = 32$ and LAN setting.

Table 3. Performance results of our PSI protocol.

	Comm [MB]			LAN [ms]			WAN [ms]		
	$\ell = 32$	$\ell = 48$	$\ell = 64$	$\ell = 32$	$\ell = 48$	$\ell = 64$	$\ell = 32$	$\ell = 48$	$\ell = 64$
$n = 2^8$	2.4	2.8	3.5	1407	1498	1556	5034	5730	6847
$n = 2^{10}$	9.4	10.6	13.2	2604	2710	3024	16812	18731	22976
$n = 2^{12}$	36.9	42.1	52.4	6804	7323	7891	62577	70956	87091
$n = 2^{14}$	147.4	168.0	209.3	22488	24486	27757	245277	278412	344105
$n = 2^{16}$	589.0	671.5	836.6	85134	92271	106268	975384	1107216	1370755

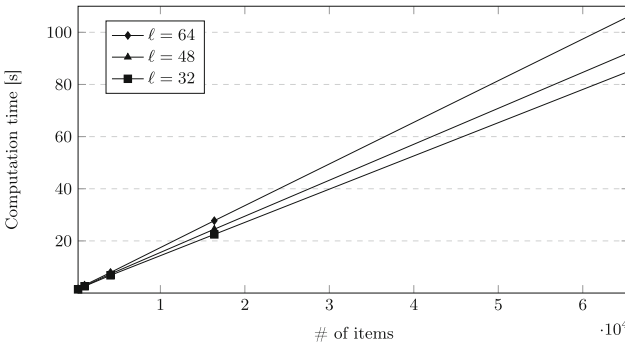


Fig. 1. Computation complexity of our PSI protocol for different set sizes and item bit lengths in the LAN setting.

Table 3 shows that for $n = 2^{12}$ and $\ell = 32$ our protocol's communication and computation complexity is (36.9 MB, 6804 ms in LAN) while the numbers

for other circuit based PSI protocols of [26,27] and [25] respectively are (9 MB, 1199 ms), (51 MB, 5031 ms) and (130 MB, 7825 ms) as given in [26].

With Fig. 2, we compare concrete computation complexity of our PSI protocol with the complexity of no-stash PSI solution of [26] for the case that $\ell = 32$ and the setting is LAN. In practice, circuit-based solutions like [26] enjoy the benefits of recent advances in the two-party computation techniques. Therefore, we conclude that Bloom filter based solutions and oblivious transfer extension techniques should be investigated further in practice.

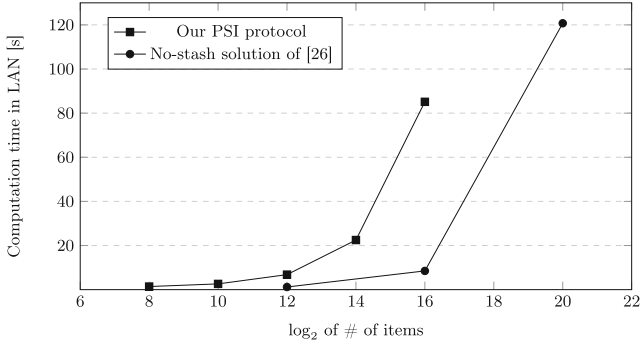


Fig. 2. Comparison of our protocol with no-stash PSI solution of [26].

8 Conclusion

We proposed the first private set intersection (PSI) protocol achieving linear communication and computation complexities while outputting a function of the membership results to be used in larger secure two-party protocols to compute other functionalities over the intersection set. To construct such a protocol, we first used one-time oblivious programmable pseudo-random function (OPPRF) based on existing Bloom filter based PSI solutions and then proposed a private set membership (PSM) protocol. To reduce the complexity while converting the PSM solution to a PSI protocol using hashing techniques, we constructed another primitive that is called a batch one-time OPPRF. Finally, using these new constructions, we introduced our PSI protocol with linear communication and computation complexities. We also implemented our protocols to validate our performance analysis and show concrete efficiency of our protocols. We leave security against malicious adversaries, and multi-party PSI with bi-oblivious data transfer as future work.

Acknowledgements. We acknowledge support from TÜBİTAK, the Scientific and Technological Research Council of Turkey, under project number 119E088. The authors thank Sherman Chow for his valuable comments.

References

1. Beaver, D.: Correlated pseudorandomness and the complexity of private computations. In: ACM STOC (1996)
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* **13**(7), 422–426 (1970)
3. Boyle, E., et al.: Efficient two-round OT extension and silent non-interactive secure computation. In: ACM CCS (2019)
4. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudo-random correlation generators: silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 489–518. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_16
5. Ciampi, M., Orlandi, C.: Combining private set-intersection with secure two-party computation. In: Catalano, D., De Prisco, R. (eds.) SCN 2018. LNCS, vol. 11035, pp. 464–482. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98113-0_25
6. De Cristofaro, E., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: Pieprzyk, J., Sadeghi, A.-R., Manulis, M. (eds.) CANS 2012. LNCS, vol. 7712, pp. 218–231. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35404-5_17
7. De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 143–159. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_13
8. Davidson, A., Cid, C.: An efficient toolkit for computing private set operations. In: Pieprzyk, J., Suriadi, S. (eds.) ACISP 2017. LNCS, vol. 10343, pp. 261–278. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59870-3_15
9. Debnath, S.K., Dutta, R.: Secure and efficient private set intersection cardinality using bloom filter. In: Lopez, J., Mitchell, C.J. (eds.) ISC 2015. LNCS, vol. 9290, pp. 209–226. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23318-5_12
10. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: ACM CCS (2013)
11. Falk, B.H., Noble, D., Ostrovsky, R.: Private set intersection with linear communication from general assumptions. In: ACM WPES (2019)
12. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_17
13. Ghosh, S., Nilges, T.: An algebraic approach to maliciously secure private set intersection. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 154–185. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_6
14. Huang, Y., Evans, D., Katz, J.: Private set intersection: are garbled circuits better than custom protocols? In: NDSS (2012)
15. Ion, M., et al.: On deploying secure computing commercially: private intersection-sum protocols and their business applications. In: IEEE Euro S&P (2020)
16. Ion, M., et al.: Private intersection-sum protocol with applications to attributing aggregate ad conversions. *IACR Cryptol. ePrint Arch.* **2017**, 738 (2017)
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. Karakoç, F., Nateghizad, M., Erkin, Z.: SET-OT: a secure equality testing protocol based on oblivious transfer. In: ARES (2019)

19. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: ACM CCS (2016)
20. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. In: ACM CCS (2017)
21. Lindell, Y.: How to simulate it – a tutorial on the simulation proof technique. In: Lindell, Y. (ed.) *Tutorials on the Foundations of Cryptography*. ISC, pp. 277–346. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57048-8_6
22. Meadows, C.A.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: IEEE S&P (1986)
23. Pagh, R., Rodler, F.F.: Cuckoo hashing. *J. Algorithms* **51**(2), 122–144 (2004)
24. 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
25. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: private set intersection using permutation-based hashing. In: *USENIX Security* (2015)
26. 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
27. 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
28. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: *USENIX Security* (2014)
29. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.* **21**(2), 7:1–7:35 (2018)
30. Rabin, M.O.: How to exchange secrets by oblivious transfer. Technical report, Harvard Aiken Computation Laboratory Technical Report TR-81 (1981)
31. Shamir, A.: On the power of commutativity in cryptography. In: de Bakker, J., van Leeuwen, J. (eds.) *ICALP 1980*. LNCS, vol. 85, pp. 582–595. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10003-2_100
32. Zhao, Y., Chow, S.S.M.: Are you the one to share? Secret transfer with access structure. *Proc. Priv. Enhancing Technol.* **2017**(1), 149–169 (2017)
33. Zhao, Y., Chow, S.S.M.: Can you find the one for me? In: *ACM WPES* (2018)



Compact Multi-Party Confidential Transactions

Jayamine Alupotha¹(✉), Xavier Boyen¹, and Ernest Foo²

¹ Queensland University of Technology, Brisbane, Australia
alupotha@qut.edu.au, xb@boyen.org

² Griffith University, Brisbane, Australia
e.foo@griffith.edu.au

Abstract. “Confidential Transactions”, integrated transactions of commitments, signatures, and zero-knowledge range proofs, are favored for their ability to hide transaction amounts. In the real world, multi-party fund transfers are highly desirable for personal and business security. Unfortunately, existing *unproven* Multi-Party Confidential Transactions are linear in the (exact) number of co-owners; hence they are not compact, very scalable, nor private (leak number of users and their public information). In this study, we provide provably secure *private, compact Multi-Party Confidential Transactions*, in both the “unanimous” N -out-of- N and “threshold” T -out-of- N settings. Unlike other schemes, our multi-party transactions have the size of single-owner transactions and hide the number of participants. To the best of our knowledge, ours is the first proven secure multi-party and threshold confidential transaction protocol.

1 Introduction

Privacy of cash systems has been an academically interesting topic since well before the blockchain [16]; however, it is the latter that has spurred the study of cash privacy in a completely decentralized setting. Early cryptocurrencies like Bitcoin and Ethereum [26], provide privacy for senders and recipients but do not hide the transaction amount by design. This problem was solved by Gregory Maxwell’s “Confidential Transactions” (CT) [12]. An interesting alternative approach to privacy (and efficiency) was proposed in a 2016 cash system by Tom Elvis Jedusor called Mumblewimble [10] improving the idea of CT. The distinctive property of Mumblewimble over other privacy-preserving cryptocurrencies is that it “compresses” the transaction history by trimming the details of all consumed transactions without affecting the verifiability of the ledger. Mumblewimble was quickly followed by a security overview of the same [18], and generalized to the notion of aggregate cash systems [8] with game-based security proofs.

At a higher level, we explain Maxwell’s CT as follows. An asset (coin bundle) of a confidential cash system¹ is a Pedersen commitment [17] $C_{v,k}$ with v number of coins and a blinding key k . The idea behind a commitment is once a commitment is immutably published, the coin amount or the blinding key can not be changed; hence, it is hard to find another coin amount or another blinding key that gives the same commitment. Also, Pedersen commitments hide the value leading to conceal the coin amount.

Owning coins in confidential cash systems means the owner knows the secret key k of the published commitment. When spending, the owner hands over the secret key to the new owner by sending a pre-transaction $ptx = (v, k)$. Then the new owner creates a new commitment $C_{v,k'}$ with a new secret key k' and publishes the transaction $(C_{v,k}, C_{v,k'}, \sigma_{k'-k}(E, \varepsilon))$ with a digital signature $\sigma_{k'-k}(E, \varepsilon)$ of signing-verification key pair $(k' - k, E)$ on an empty message ε . Due to the additive homomorphic Pedersen commitments, CT can use excess value or the difference of output coin bundles and input coin bundles as the verification key E for the signature. In the above example, $E = C_{v,k'} \cdot C_{v,k}^{-1} = C_{0,k'-k}$ and verifiers check; “Is $\sigma_{k'-k}(E, \varepsilon)$ valid for (E, ε) ?”. Therefore, only the actual owner can create the transaction since only he/she knows $k' - k$. However, Pedersen commitments fail to generate accurate transactions for negative amounts and cash overflows, for example, an output coin bundle (C_{5,k_1}, C_{-3,k_2}) of input $C_{2,k}$ illegally creates coins out thin air. Therefore, CTs employ zero-knowledge range proofs $\pi_{v,k}$ to verify the range of v of commitment $C_{v,k}$ such as Borromean proofs [13, 19] and Bulletproofs [6]. The complete confidential transaction of the example discussed above is $tx = (C_{v,k}, C_{v,k'}, (\pi_{v,k'}, E, \sigma(E, \varepsilon)))$.

Assume the following example that Alice wants to send ρ coins to Bob when she has v coins in a coin bundle $C_{v,k}$. Alice **sends** a transaction creating two coin bundles $(C_{v-\rho,k_1}, C_{\rho,k_2})$ to isolate ρ coins from v coins leaving her with $v - \rho$ coin balance. Then she sends a pre-transaction (ρ, k_2) to Bob. Bob **receives** coins by changing the key of the coin bundle to k' . As Alice does not know k' , only Bob can spend ρ coins. Therefore, the entire fund transferring process has a **pre-transaction** and two transactions; the **sending transaction** and the **receiving transaction**.

An intriguing question in online cash systems is how to handle funds belonging to more than one owner, and to control or swap ownership [1]. In cryptocurrencies, the ledger itself should provide this functionality as it would be cumbersome to require the parties to meet physically to sign a multi-party transaction, and no assistance will come from a bank teller or trusted third party. On the other hand, the ledger must provide security against an “insider” attack, where a co-owner tries to claim jointly owned funds by creating a spending transaction without the consent of the other co-owners.

¹ Note that we do not specify how the asset details are recorded in the cash system, meaning that the asset details may be permanent like in Bitcoin blockchain or aggregatable in Mimblewimble variants. The confidential transaction protocol is compatible with any secure cash system, which prevents double spending if the unspent assets are accessible.

Multi-party transactions have two primary use cases: to handle funds belonging to businesses and to increase the *accessibility, availability, and security* of personal funds with multiple wallets^{2,3}. An organization may implement a policy that the funds must be spent with the consent of a majority of the directors. An individual might prefer having wallets acts as separate co-owners across different devices where one wallet is sufficient to access the funds. Similar manner, due to heightening threats ranging from *ransomware attacks*, someone may prefer to spread their wallets over multiple devices to make sure even if one wallet is under an attack, other wallets will grant access to the assets. A related recommendation is to secure funds with multiple secret keys stored in different locations to make the funds harder to steal.

In this study, our goal is to build a compact, private, Multi-Party Confidential Transactions (MCT),

- Compact MCT—MCTs are as short as regular single-owner transactions; in other words, the multi-party commitments, signatures, and range-proofs are *indistinguishable* from the single-party commitments, signatures, and range-proofs. A surprising fact showed in [14] is compacting multi-signatures saves ~ 35 GB from the Bitcoin blockchain in 2018. A similar size reduction can be achieved with compact MCT if we apply the same statistics.
- Private MCT—the very fact that multiple parties are involved (and their number), is hidden. The objective is to hide the number of required secret keys and their public information from the attackers who try to steal funds or blackmail owners with ransomware attacks.

Consider the following native approach: Alice sets a coin bundle as $C = C_{v, k_1+k_2}$ and sends pre-transactions; (C_{v, k_1+k_2}, v, k_1) to Bob and (C_{v, k_1+k_2}, v, k_2) to Charles, intending that consent from both co-owners be required to spend the coins since neither one will know k_1+k_2 . Bob and Charles generate respective keys k'_1, k'_2 at random and shares partial commitments as $C_{\text{Bob}} = C_{v/2, k'_1}$ and $C_{\text{Charles}} = C_{v/2, k'_2}$ aiming to create a new coin bundle C' as,

$$C' = C_{\text{Bob}} \cdot C_{\text{Charles}} = C_{v, k'_1+k'_2}.$$

However, Bob waits for Charles's partial transaction $C_{v/2, k'_2}$ and shares a cheating partial commitment $C_{\text{Bob}} = C_{v/2, k'_1} \cdot (C_{v/2, k'_2})^{-1}$ with Charles. Now, C' will be,

$$\text{Rogue Commitment} : C' = C_{\text{Bob}} \cdot C_{\text{Charles}} = C_{v, k'_1}$$

not $C_{v, k'_1+k'_2}$. Now, Bob and Alice can spend coins without Charles's permission since they have all the keys that need to create a valid transaction. For example, if Alice is a buyer of Bob and Charles, the product will not be shipped until both Bob and Charles take the ownership of Alice's coins by changing the keys. With the attack, Charles will decide to send the product to Alice, thinking that he is a co-owner.

² A wallet is an application that securely stores secret keys. Generally, wallets are password protected.

³ Multiple wallets with different keys replicate the shadow co-owners of the same owner.

1.1 Our Contribution

We introduce a compact, private Multi-Party Confidential Transaction protocol that is adaptable with traditional cash systems and aggregate cash systems. We offer two constructions: a simpler N -party unanimous transfer and a general T/N threshold transfer.

To defeat Rogue Key attacks in our compact MCT, we use the following commitment generation when $H()$ is a one-way hash function. First, Bob and Charles share $\mathbf{P}_{\text{Bob}} = C_{0,k'_1}$, $\mathbf{P}_{\text{Charles}} = C_{0,k'_2}$ with each other. Then both separately compute,

$$\begin{aligned} S &= \{\mathbf{P}_{\text{Bob}}, \mathbf{P}_{\text{Charles}}\} \\ C_{\text{Bob}} &= \mathbf{P}_{\text{Bob}}^{H(\mathbf{P}_{\text{Bob}}, S)} \cdot C_{v/2,0} = C_{v/2, H(\mathbf{P}_{\text{Bob}}, S)k'_1} \\ C_{\text{Charles}} &= \mathbf{P}_{\text{Charles}}^{H(\mathbf{P}_{\text{Charles}}, S)} \cdot C_{v/2,0} = C_{v/2, H(\mathbf{P}_{\text{Charles}}, S)k'_2} \\ C' &= C_{\text{Bob}} \cdot C_{\text{Charles}} = C_{v, [H(\mathbf{P}_{\text{Bob}}, S)k'_1 + H(\mathbf{P}_{\text{Charles}}, S)k'_2]}. \end{aligned}$$

This method prevents Bob from executing the previous Rogue Key attack. Still, Bob can run $C' \cdot (\mathbf{P}_{\text{Charles}}^{H(\mathbf{P}_{\text{Charles}}, S)})^{-1}$ on the transaction. Therefore, our confidential transactions sign $E = C' \cdot C^{-1}$ instead of an empty message, committing C' to the transaction. With a secure signature scheme and range proof scheme, our MCT protocol prevents Rogue Key attacks. If these schemes are compact, our MCTs are indistinguishable from single-owner transactions. We provide the detailed MCT protocol in Sect. 3.

We emphasize **provable security** as one of our contributions. Amongst cryptocurrencies, [21] identified a denial-of-spending attack on Zerocoin, which has the academic lineage and was heretofore believed secure. In fact, the original Mimblewimble was similarly broken in [8]. As this and other examples show, a thorough cryptographic investigation is required to prevent similar incidents in MCT. Therefore, we provide the formal security model of MCT in Sect. 4.1 precisely defining the required security properties of the signature scheme and range proof scheme. Then we propose *proven secure*, compact, Multi-Party Bulletproof Range Proofs (MBP) in Sect. 4.3. Finally, to show the practicality of our MCT, we prove the security of MCT with Schnorr signatures [14, 22], BLS signatures [3, 4], and MBP in Sect. 4.

1.2 Related Work

In Table 1, we compare our MCT protocol with the existing CT protocols, including Mimblewimble variants. While other CT protocols do not support MCTs, one online report [25] suggests a MCT for Grin [24] with Schnorr signatures [22]. However, these MCTs are *neither compact nor private*, as they sign the vector of commitments from all n co-owners, that leaks their number n and has size $\Theta(n)$. More importantly, they *do not provide any security proofs* regarding how they prevent Rogue Key attacks.

Table 1. A Comparison of Confidential Transaction Protocols

CT Protocol	Signature	Range Proofs	Security Proofs	MCT	Private MCT	Compact MCT
[12]	–	Borr. [13]	✗	✗	✗	✗
[10]	–	Borr. [13]	✗	✗	✗	✗
[18]	Sinking Sig	Borr. [13]	✓	✗	✗	✗
[8]	Sch./BLS	–	✓	✗	✗	✗
Grin [24]	Sch. [22]	MBP [6]	–	✗	✗	✗
[25]	Sch. [22]	MBP [25]	✗	✓	✗	✗
Beam [2]	Sch. [22]	BP [6]	–	✗	✗	✗
Ours	Sch./BLS [3, 14]	MBP (this paper)	✓	✓	✓	✓

Multi-party Bulletproofs (MBP). Older range proofs, e.g., Borromean range proofs [13] and modified Borromean range proofs [19], were linear in size, whereas the more recent Bulletproofs introduced in [6] which uses the Improved Inner-Product argument (IP)[5] can be made logarithmic, which significantly reduces the size of the entire transaction (from about 3 kB to 0.8 kB in practical implementations). MBP [7, 25] are extensions over the original Bulletproofs, where multiple owners can generate a single range proof without disclosing secret information to each other or the combiner (dealer). Our proven secure MBP is similar to [6, 7, 25] but with the following benefits.

Table 2. A Comparison of Multi-party Bulletproof Rangeproof Protocols

BP Scheme	Multi-party	Honest Combiner	Security Proofs	Compact/Private	Communications	Non-malleable
[6]	✗	–	✓	–	–	✗
[7]	✓	Not specified	✗	✗	3	✓
MBP [25]	✓	Not specified	✗	✓	2	Not specified
Our MBP	✓	No need	✓	✓	2	✓

Compact Multi-Signature Schemes. Multi-signatures have been an intensely scrutinized topic over the last decades. Many signature schemes, such as Schnorr and BLS signatures, support direct signature aggregation due to key-homomorphic properties. However, rogue key attacks [9, 11, 15, 20] make secure aggregation harder for multi-signatures, lest an attacker inserts a malicious public key—in our case, commitment—allowing him to create a complete signature—or proof—all by himself. The first compact multi-signature scheme to resist rogue key attacks without a trusted setup was proposed for Schnorr signatures in [14]. Later, [3] extended the same methodology for BLS signatures. The main advantages of compact multi-signatures such as those are that a multi-signature is indistinguishable from a signature created by a single signer, preserving privacy,

and reducing size and verification time. We adopt a similar methodology as used in compact multi-signatures [3, 14] to generate MCT commitments in our scheme and modify both the signatures and the multi-party Bulletproofs in tandem.

2 Preliminaries

Notations. The notation used in the rest of the paper is as follows. Bold letters such as \mathbf{a} denote arrays and \mathbf{a}_i is the i th element of the array while $\mathbf{a} = [\mathbf{a}_i]_{i=1}^n$ is an array with n elements. Also, $|\mathbf{a}|$ denotes the number of elements in the array. \mathbf{A}^T of a 2×2 matrix \mathbf{A} denotes the transpose matrix. The letter b is reserved for Boolean values, which can be $0 = False$ or $1 = True$ unless stated otherwise. Given S , a finite non-empty set, $s \stackrel{\$}{\leftarrow} S$ denotes the assignment of s as a uniformly random element of S . \wedge and \vee are the *and* and *or* logical operators. ‘ \parallel ’ denotes vector concatenation, for example, $\{a, b\} \parallel \{c\} = \{a, b, c\}$. For a cyclic group $\mathbb{G} = \langle g \rangle$, g denotes a generator of the group \mathbb{G} . We use subscript indexing such as a_i to denotes the i th element of the vector \mathbf{a} . We use λ for the security parameter, often omitted when clear from context. Let $\epsilon(\lambda)$ is a negligible function such that $\epsilon(\lambda) = O(1/\lambda^c)$ for every $c \in \mathbb{N}$.

Definition 1 (Discrete Log Problem). For a group $\mathbb{G} = \langle g \rangle$ of prime order p , $\text{Adv}_{\mathbb{G}}^{DL}$ for an adversary \mathcal{A} is defined as, $\text{Adv}_{\mathbb{G}, \mathcal{A}}^{DL} := Pr[y \stackrel{?}{=} g^x \mid y \stackrel{\$}{\leftarrow} \mathbb{G}, x \stackrel{\$}{\leftarrow} \mathcal{A}(y)]$. *DL problem is (τ, e) -hard if $\mathcal{A}(\tau, e)$ runs it at most τ times and $\text{Adv}_{\mathbb{G}, \mathcal{A}}^{DL} \leq e$.*

Definition 2 (Computational ψ -co-Diffie-Hellman Problem). Let group $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ of prime order p , and $\mathcal{O}^\psi(\cdot)$ is an oracle which returns $g_1^\alpha \in \mathbb{G}_1$ given $g_2^\beta \in \mathbb{G}_2$. $\text{Adv}_{\mathbb{G}_1, \mathbb{G}_2}^{\psi\text{-co-CDH}}$ for an adversary \mathcal{A} is defined as, $\text{Adv}_{\mathbb{G}_1, \mathbb{G}_2, \mathcal{A}}^{\psi\text{-co-CDH}} := Pr[y \stackrel{?}{=} g_1^{\alpha\beta} \mid (\alpha, \beta) \stackrel{\$}{\leftarrow} \mathbb{Z}_p^2, y \stackrel{\$}{\leftarrow} \mathcal{A}(g_1^\alpha, g_1^\beta, g_2^\beta)]$. *ψ -co-CDH problem is (τ, e) -hard if $\mathcal{A}(\tau, e)$ runs it at most τ times and $\text{Adv}_{\mathbb{G}_1, \mathbb{G}_2, \mathcal{A}}^{\psi\text{-co-CDH}} \leq e$.*

2.1 Homomorphic Pedersen Commitment Scheme

Let CM be the Pedersen commitment scheme [17], defined as follows,

- $\text{CM.Set}(\lambda) : pp = (\mathbb{G}, g, h, p) //$ where $\mathbb{G} = \langle g \rangle = \langle h \rangle$ is a group of prime order $p \in \{0, 1\}^\lambda$, and the discrete logarithms of g and h relative to each other are unknown — they are “nothing-up-my-sleeve” (NUMS) group generators.
- $\text{CM.Cmt}(pp, v, k) : C_{v,k} = g^k h^v //$ commits value v , key k
- $\text{CM.Ver}(pp, C, v, k) : C \stackrel{?}{=} g^k h^v //$ verifies commitment C

Pedersen commitments hold the following security properties.

Definition 3 (Completeness). When $pp \leftarrow \text{CM.Set}(\lambda)$, $v \in \mathcal{V}_{pp}$, $r \in \mathbb{Z}_p$, and $C \in \mathcal{C}_{pp}$, CM for any λ is complete if $\text{CM.Ver}(pp, \text{CM.Cmt}(pp, v, r), v, r) = True$.

Definition 4 (Hiding and Binding). When $pp \leftarrow \text{CM.Set}(\lambda)$, CM is hiding and binding for any p.p.t. adversary \mathcal{A} if, $\text{Adv}_{\text{CM}}^{\text{HID}} = |\text{Pr}[\text{Game}_{\text{CM},\mathcal{A}}^{\text{HD}}(pp)] - 1/2| \leq \epsilon(\lambda)$, $\text{Adv}_{\text{CM}}^{\text{BD}} = \text{Pr}[\text{Game}_{\mathbb{G},\mathcal{A}}^{\text{BD}}(pp)] \leq \epsilon(\lambda)$.

$$\begin{array}{ll}
 \text{Game}_{\text{CM},\mathcal{A}}^{\text{HD}}(pp) : // \text{ value range is } \mathcal{V}_{pp} & \text{Game}_{\text{CM},\mathcal{A}}^{\text{BD}}(pp) : \\
 (v_0, v_1) \xleftarrow{\$} \mathcal{V}_{pp}, b \xleftarrow{\$} [0, 1], & (v_0, k_0, v_1, k_1) \leftarrow \mathcal{A}(pp) \\
 C \leftarrow \text{CM.Cmt}(pp, v_b, k \xleftarrow{\$} \mathbb{Z}_p) & \text{return } v_0 \stackrel{?}{\neq} v_1 \wedge \text{CM.Cmt}(pp, v_0, k_0) \stackrel{?}{=} \\
 b' \leftarrow \mathcal{A}(pp, C), \text{ return } (b' \stackrel{?}{=} b) & \text{CM.Cmt}(pp, v_1, k_1)
 \end{array}$$

Theorem 1. Pedersen commitments are complete, perfectly hiding, and computationally binding (prefer [17]).

Homomorphism. The additive homomorphism of Pedersen commitments preserves the arithmetic operation “addition” throughout the commitments as, $\text{CM.Cmt}(pp, v_0, r_0) \cdot \text{CM.Cmt}(pp, v_1, r_1) = \text{CM.Cmt}(pp, v_0 + v_1, r_0 + r_1)$ ($\in \mathbb{G}$).

2.2 Compact Multi Signature Scheme

Here, we define a generic compact multi-signature scheme SIG (influenced by compact Schnorr and BLS multi signatures [3, 14]) suitable for MCT. Here, each co-signer i holds the secret key sk_i **without** sharing them with any other co-signer.

- $\text{SIG.Set}(\lambda) : pp //$ generates public parameters for λ
- $\text{SIG.KG}(pp, n) : [(sk_i, pk_i, \bar{P})]_{i=1}^n //$ Co-signer i generates partial signing and verification keys (sk_i, pk_i) . Then cosigners combine verification keys to generate the aggregate verification key \bar{P} which is identical to a normal verification key.
- $\text{SIG.Sign}(pp, [sk_i]_{i=1}^n, m, n) : \sigma //$ Co-signers collaboratively compute the aggregate multi-signature on message m without revealing the signing keys. The aggregate multi-signature is indistinguishable from a normal signature.
- $\text{SIG.Ver}(pp, pk, m, \sigma) : \text{True/False} //$ bulk verification of $|\sigma|$ number of signatures when $|pk| = |m| = |\sigma|$. Hence the aggregate verification keys and aggregate signatures are indistinguishable; the same verification function is used for both signatures and aggregate multi-signatures.

The completeness and the existential unforgeability under the chosen message attack (EUF-CMA) of the signature schemes are defined below.

Definition 5 (Completeness of Signatures). When $pp \leftarrow \text{SIG.Set}(\lambda)$ for any λ and $m \xleftarrow{\$} \mathcal{M}_{pp}$, SIG is complete if,

$$\left[\text{SIG.Ver}(pp, \bar{P}, m, \sigma) \mid \begin{array}{l} [(sk_i, pk_i, \bar{P})]_{i=1}^n \leftarrow \text{SIG.KG}(pp, n), \\ \sigma \leftarrow \text{SIG.Sign}(pp, [sk_i]_{i=1}^n, m, n) \end{array} \right] = \text{True}$$

Definition 6 (EUF-CMA). Assume an instance of SIG when $pp \leftarrow \text{SIG.Set}(\lambda)$ for any λ . SIG is EUF-CMA if, $\text{Adv}_{\text{SIG},n,\mathcal{A}}^{\text{EUF-CMA}} = \Pr[\text{Game}_{\text{SIG},n,\mathcal{A}}^{\text{EUF-CMA}}(pp)] \leq \epsilon(\lambda)$.

$$\begin{array}{ll} \text{Game}_{\text{SIG},n,\mathcal{A}}^{\text{EUF-CMA}}(pp) : & \text{Oracle Sign}_{sk}(m) : \\ [(\mathbf{sk}_i, \mathbf{pk}_i, \overline{P})]_{i=1}^n \leftarrow \text{SIG.KG}(pp, n) & \sigma \leftarrow \text{SIG.Sign}(pp, sk, m) \\ (m', \sigma') \leftarrow \mathcal{A}^{\text{Sign}_{sk_1}(\cdot)}(pp, [\mathbf{sk}_i]_{i=2}^n, \mathbf{pk}_1) & \mathbf{Q} = \mathbf{Q} \cup \{m, \sigma\}, \text{ return } \sigma \\ \text{return } (m', \sigma') \notin \mathbf{Q} \wedge \text{SIG.Ver}(pp, \overline{P}, m', \sigma') & \end{array}$$

2.3 Non-interactive Zero-Knowledge Compact Multi-party Range Proofs

Zero-knowledge range proofs prove that a hidden value in a commitment is in the accepted range without revealing the value, ex: v is in $[0, 2^{64}]$. We define a generic compact, multi-party range proof scheme suitable for CT.

Let RP is a compact range proof scheme with the following functions,

- $\text{RP.Set}(\lambda) : pp //$ generates public parameters for λ
- $\text{RP.Priv}(pp, C_{v, \sum \mathbf{k}}, v, \mathbf{k}, n) : \pi //$ Co-provers collaboratively compute a compact range proof for the commitment $C_{v, \sum \mathbf{k}}$ when each co-prover i has unshared \mathbf{k}_i .
- $\text{RP.Ver}(pp, C, \pi) : \text{True/False} //$ bulk verification for $|\pi|$ number of range proofs when $|C| = |\pi|$.

We define the completeness, soundness, zero-knowledge, non-malleability, and security against honest-but-curious combiners of range proofs below.

Definition 7 (Completeness). For any p.p.t. adversary \mathcal{A} for any λ , RP is complete if, $\left[\text{RP.Ver}(pp, C, \text{RP.Priv}(pp, C, v, \mathbf{r}, n)) \mid \mathbf{r} \xleftarrow{\$} \mathbb{Z}_p^n, v \xleftarrow{\$} \mathcal{V}_{pp}, C \leftarrow \text{CM.Cmt}(pp, v, \sum \mathbf{r}) \right]$.

The zero-knowledge property indicates that the verifier learns nothing besides the statement being proven. Here, we consider an honest verifier who chooses his/her challenge parameters at random and independently from the prover's messages, which is called the public coin argument.

Definition 8 (Honest Verifier Zero-Knowledge). RP is public coin Honest Verifier Zero-Knowledge for \mathcal{R} if there exists a p.p.t. simulator \mathcal{S}_{TR} such that, $\text{Adv}_{\text{RP},\mathcal{A}}^{\text{ZK}} = \epsilon(\lambda)$ when $\text{Adv}_{\text{RP},\mathcal{A},N}^{\text{ZK}}$ is,

$$\left| \Pr \left[\text{RP.Ver}(pp, C, \pi) \mid pp \leftarrow \text{RP.Set}(\lambda), \mathbf{r} \xleftarrow{\$} \mathbb{Z}_p^n, v \leftarrow \mathcal{A}'(pp), v \in \mathcal{V}_{pp} \right] - \Pr \left[\text{RP.Ver}(pp, C, \pi) \wedge \mathcal{A}(pp, C, \pi) = 1 \mid C \leftarrow \text{CM.Cmt}(pp, v, \sum \mathbf{r}), \pi \leftarrow \text{RP.Priv}(pp, C, v, \mathbf{r}) \right] \right|$$

Here, $\mathcal{A}(tr) = b$ denotes whether \mathcal{A} accepts the transcript or not, and $v \leftarrow \mathcal{A}'(pp)$, $v \in \mathcal{V}_{pp}$ indicates that the adversary gets to choose the distribution of the (valid) witness. Without knowing the witness or its distribution, the simulator produces a distribution of proof transcripts (pp, C) that should be indistinguishable from a honestly generated proof. If the verifier can not make that distinction, he or she will have learned nothing about the witness.

The soundness of a proof system requires that it be infeasible (for arguments) or impossible (for proofs) to create a proof of a false statement. This is shown by exhibiting an extractor algorithm χ , which either extracts a valid witness (or, for arguments, breaks a hardness assumption) by interacting with the prover. The computational knowledge soundness property defines that no p.p.t. adversary can create a valid proof without χ being able to extract a valid witness.

Definition 9 (Computational Knowledge Soundness (KS)). *RP is computationally KS if, $\text{Adv}_{RP, \mathcal{A}}^{KS} = \Pr \left[\chi^{\mathcal{E}(\cdot)} \notin \mathcal{V}_{pp} \mid pp \leftarrow \text{RP.Set}(\lambda), (C, \theta) \leftarrow \mathcal{A}_1(pp) \right] = \epsilon(\lambda)$.*

Oracle $\mathcal{E}(\varphi)$: $\pi \leftarrow \mathcal{A}_2(C, \theta, \varphi)$, **if** $\text{RP.Ver}(pp, C, \pi)$ **return** π **else abort**

Confidential Transactions use homomorphic commitments to compress the transaction history where the malleability of the commitments is used for its advantage. However, Non-Malleability⁴ of range proofs is an *essential property* which prevents an adversary from creating a fresh range proof using another given range proof while creating a new commitment. (The impact of malleability on CTs is discussed under Lemma 9).

Definition 10 (Non-malleability). *When $pp \leftarrow \text{RP.Set}(\lambda)$ for any λ , RP is a non-malleable range proof system if, $\text{Adv}_{RP, \mathcal{A}}^{NM} = \Pr [\text{Game}_{CT, \mathcal{A}}^{NM}(pp)] \leq \epsilon(\lambda)$. For Pedersen commitments, $\text{CM.Mal}(C, k') := C \cdot g^{k'}$.*

$\text{Game}_{CT, \mathcal{A}}^{NM}(pp)$: $(C, \pi) \leftarrow \mathcal{A}_1(pp)$, $(\pi', k') \leftarrow \mathcal{A}_2(pp, C, \pi)$
 $C' \leftarrow \text{CM.Mal}(C, k')$, **return** $k' \neq 0 \wedge \text{RP.Ver}(pp, C', \pi')$

Since multi-party range proofs are generated with the help of a combiner, we define the following security property against ‘‘Honest-but-Curious’’ combiners. The honest-but-curious combiners generate valid proofs for valid values, but if there is an attack that they can mimic a co-owner to steal funds, they execute the attack.

Definition 11 (Insider Security). *For any λ , RP is secure against honest-but-curious combiners if, $\text{Adv}_{PR, \mathcal{A}}^{HC} = \Pr [\text{Game}_{PR, \mathcal{A}}^{HC}(pp)] \leq \epsilon(\lambda)$.*

⁴ This property is an additional property that is overlooked by the original Bulletproofs range proofs [6].

$\underline{\text{Game}_{PR,A}^{HC}(pp)} :$ $\mathbf{k} \stackrel{\$}{\leftarrow} \mathbb{Z}_p^n, v \stackrel{\$}{\leftarrow} \mathcal{V}_{pp}$ $(v, C, \pi) \leftarrow \mathcal{A}^{\text{Prv}_{k_n}}(pp, v, g^{\sum \mathbf{k}}, \mathbf{k} \setminus \{\mathbf{k}_n\})$ $\text{return } C = g^{\sum \mathbf{k}} h^v \wedge (C, \pi) \stackrel{?}{\notin} \Pi$	$\underline{\text{Oracle Prv}_{k_n}(pp, v, C, g^{\sum \mathbf{k}}, \varphi)} :$ $\text{if } C = g^{\sum \mathbf{k}} h^v \neg \text{PR.Ver}_{\varphi}(v, C, \varphi) :$ $\quad \text{return } \perp$ $\theta \stackrel{\$}{\leftarrow} \text{PR.Prv}_{\varphi}(v, \mathbf{k}_n, C, \varphi)$ $\text{if RP.Ver}(pp, C, \theta = \pi) : \Pi \cup (C, \pi)$ $\text{return } \theta$
---	---

3 Compact Multi-party Confidential Transactions

In this section, we present the compact, private MCT protocol and the security model for both N -party unanimous and T/N -threshold transactions. We start by defining the basic protocols required for MCTs. Note that unlike Maxwell's CT which signs empty messages, MCT protocol signs the excess value E .

```
// generates public parameters
MCT.Set( $\lambda$ ) := ({ppCM = CM.Set( $\lambda$ ),
ppSIG = SIG.Set( $\lambda$ ), ppRP = RP.Set( $\lambda$ )})
```

```
// generates single-party CT
-when input asset details are ( $v, \mathbf{k}$ )
-and output asset details are ( $v', \mathbf{k}'$ ).
```

```
MCT.Tx( $pp, v, \mathbf{k}, v', \mathbf{k}'$ )
if  $|v| \neq |\mathbf{k}| \vee |v'| \neq |\mathbf{k}'| \vee$ 
 $\sum v \neq \sum v'$  : return  $\perp$ 
 $C \leftarrow \text{MCT.Coin}(pp, v, \mathbf{k})$ 
 $C' \leftarrow \text{MCT.Coin}(pp, v', \mathbf{k}')$ 
 $E := \prod C' \cdot (\prod C)^{-1}$ 
 $\sigma \leftarrow \text{SIG.Sign}(pp, \sum \mathbf{k}' - \sum \mathbf{k}, E, n)$ 
 $\pi' \leftarrow [\text{RP.Prv}(pp, C'_i, v'_i, \mathbf{k}'_i)]_{i \in [1, |v'|]}$ 
return  $tx := (C, C', K = (\pi', E, \sigma))$ 
```

```
// combines partial transactions  $tx$ 
-with final signatures and range proofs
```

```
MCT.Combine( $pp, \sigma, \pi, tx, w_v, w_{v'}$ ):
 $C := [\prod_{j=1}^{|tx|} tx_j.C_i]_{i=1}^{w_v}$ 
 $C' := [\prod_{j=1}^{|tx|} tx_j.C'_i]_{i=1}^{w_{v'}}$ 
return  $tx = (C, C', (\pi, \prod_{j=1}^{|tx|} tx_j.E, \sigma))$ 
```

```
// verifies a confidential transaction  $tx$ 
MCT.TxVer( $pp, tx$ ) :
( $C, C', K = (\pi, E, \sigma)$ ) :=  $tx$ 
return  $\text{RP.Ver}(pp, C', \pi) \wedge$ 
 $(\prod E \stackrel{?}{=} \prod C' \cdot (\prod C)^{-1}) \wedge$ 
 $\text{SIG.Ver}(pp, E, E, \sigma)$ 
```

```
// generates a partial transaction
-without signatures and range proofs
-when input asset details are ( $v, \mathbf{k}$ )
-and output asset details are ( $v', \mathbf{k}'$ ).
```

```
MCT.ParTx( $pp, v, \mathbf{k}, v', \mathbf{k}'$ ) :
if  $|v| \neq |\mathbf{k}| \vee |v'| \neq |\mathbf{k}'| \vee$ 
 $\sum v \neq \sum v'$  : return  $\perp$ 
 $C \leftarrow \text{MCT.Coin}(pp, v, \mathbf{k})$ 
 $C' \leftarrow \text{MCT.Coin}(pp, v', \mathbf{k}')$ 
return  $tx := (C, C', K = ((),$ 
 $E := \prod C' \cdot (\prod C)^{-1}, ()))$ 
```

```
// generates commitment arrays
```

```
MCT.Coin( $pp, v, \mathbf{k}$ ) :
if  $|v| \neq |\mathbf{k}| \wedge v \notin [0, v_{max}]$  : return  $\perp$ 
return  $[\text{CM.Cmt}(pp, v_i, \mathbf{k}_i)]_{i=1}^{|v|}$ 
```

Rogue Key Attack Resistant Commitment Generation. To defeat Rogue Commitment attack, we use the key generation of [3, 14], as explained below.

	$\overline{\text{MCT.KG}(pp, n)}$
For single-party transactions	if $n \stackrel{?}{=} \text{empty}$: return
	$(sk \stackrel{\$}{\leftarrow} \mathbb{Z}_p, pk \leftarrow g^{sk}, \bar{P} = pk)$
Each co-owner j generates primary keys.	$(k_j \stackrel{\$}{\leftarrow} \mathbb{Z}_p, P_j \leftarrow g^{k_j})$
Each co-owner j shares primary public keys. // after receiving others' P_j	
Each co-owner j computes the signing keys. $S = \{P_1, \dots, P_n\}$	$sk_j \leftarrow H(P_j, S) \cdot k_j \in \mathbb{Z}_p$
Each co-owner j computes public keys.	$pk_j \leftarrow g^{sk_j}, \bar{P} \leftarrow \prod_{j=1}^n P_j^{H(P_j, S)}$
All co-owners finish key generation	return $[(sk_j, pk_j, \bar{P})]_{j=1}^n$

Note that $H()$ is a hash function such that $H : \{0, 1\}^{any} \xrightarrow{\$} \mathbb{Z}_p$. Using $\text{MCT.KG}()$, we can easily generate a secure commitment C' such that, $C' = \bar{P} \cdot h^v$. **Our aim** is to build a MCT protocol that generates the above-mentioned commitments and compatible multi-party compact signatures and multi-party compact range proofs.

Generic Multi-party Fund Transferring

Our compact, private MCT protocol works as follows. The co-owners of the asset or the coin bundle computes secret keys by themselves but shares public information with other co-owners. Finally, each co-owner generates partial transactions, and a co-owner (**combiner**) combines the partial transactions to generate the final transaction. We explain the generic sending and receiving protocols for both N -transactions and T/N -transactions below.

Here, (v, k, C) is the input coin bundles and ρ is the output coin amount array that should be sent to $|\mathbf{m}| (= |\rho|)$ groups of receivers where each group i consists of m_i receivers. The fund sending function MCT.Send isolates coins according ρ by sending the transaction tx and distributes keys as ptx for N -fund transferring or $tptx$ for T/N -fund transferring where t_i parties out of m_i receivers must agree to spend the coin bundle i . For the sake of the function-wise implementation, we input blinding key set k together. However, $k_{j,i'}$ is the partial secret key belongs to the j th co-owner for i' th input coin bundle, and the partial keys are **not** shared with anyone else.

<p>Each co-owner j, -verifies receivers' details,</p> <p>-verifies coin amount, and -generates keys for receivers.</p> <p>If there is a balance, -co-owners generate new keys.</p> <p>Each co-owner j, -sets co-receivers' signing keys, -generates partial tx, -shares partial tx with others, -computes the final E, and -verifies the excess E.</p> <p>Co-owners create the final sig. Co-owners create range proofs.</p> <p>A co-owner combines the par. txs. Each co-owner j shares pre-tx $\mathit{ptx}_{(i,l,j)}$ or openings $\mathit{tptx}_{(i,l,j)}$ with co-receiver l of the ith output.</p> <p>See Figure 1 for Shamir threshold key -distribution $\mathit{SS.Dealer}()$.</p>	<p><u>$\mathit{MCT.Send}(pp, v, k, C, \rho, n, m, \text{optional} = t)$</u> for each $j \in [1, n]$ do if $\neg(\rho = m) \vee \neg(C = v = k_j)$: return \perp if $(\sum v < \sum \rho)$: return \perp $(sk_j, pk_j, \bar{P}_j) := [\mathit{MCT.KG}(pp, m_i)]_{i \in [1, \rho']}$ // Each co-owner j share \bar{P}_j with others $k'_j := \{(\sum_{l=1}^{m_1} sk_{j,1,l}), \dots, (\sum_{l=1}^{m_{ \rho }} sk_{j, \rho ,l})\}$ if $(\sum v > \sum \rho)$: $v' := (\rho \ \{\sum v - \sum \rho\})$ $(sk', pk', \bar{P}') \leftarrow \mathit{MCT.KG}(pp, n)$ for each $j \in [1, n]$ do $k'_j := k'_j \ \{sk'_j\}$ $tx_j := \mathit{MCT.ParTx}(pp, v/n, k_j, v'/n, k'_j)$ // after receiving all partial transactions tx_j $E := \prod_{j=1}^n tx_j.E \in \mathbb{G}$ if $E \neq \prod \bar{P} \cdot \bar{P}' \cdot \prod C^{-1} \cdot C_{\sum v, 0}$: return \perp $\sigma \leftarrow \mathit{SIG.Sign}(pp, [\sum k'_j - \sum k_j]_{j=1}^n, E, n)$ $\pi := \forall i \in [1, v']$: $\mathit{RP.Priv}(pp, \prod_{j=1}^n tx_j.C'_i, v'_i, [k'_{j,i}]_{j=1}^n, n)$ $tx := \mathit{MCT.Combine}(pp, \sigma, \pi, [tx_j]_{j=1}^n, v , v')$ if $(t = \text{empty})$: // N key distribution return $tx, \mathit{ptx}_{(i,l,j)} =$ $(tx.C'_i, \rho_i, sk_{(j,i,l)}, \bar{P}_{(j,i)})_{(i,j)=(1,1,1)}^{(m_i, \rho , n)}$ else: // (T/N) key distribution $[(x_{(j,i)}, K_{(j,i)}) :=$ $\mathit{SS.Dealer}(pp, t_i, m_i, k_{(j,i)})]_{(i,j)=(1,1,1)}^{(\rho , n)}$ return $tx, \mathit{tptx}_{(i,l,j)} = (tx.C'_i,$ $\rho_i, x_{(j,i)}, K_{(j,i,l)}, \bar{P}_{(j,i)})_{(l,i,j)=(1,1,1)}^{(m_i, \rho , n)}$</p>
<p>Each co-receiver l, -generates keys,</p> <p>-generates par. tx, -shares par. txs, -computes the final excess E, and -verifies excess value.</p> <p>Co-owners create the final signature. Co-owners create range proofs.</p> <p>A co-owner combines the partial txs to generate final tx.</p>	<p><u>$\mathit{MCT.Receive}(pp, v, k, C, \bar{P}, n, m)$</u> for each $l \in [1, m]$ do if $C \neq \bar{P} \cdot \mathit{CM.Cmt}(pp, v, 0)$: return \perp $(sk', pk', \bar{P}') \leftarrow \mathit{MCT.KG}(pp, m)$ $tx_l := \mathit{MCT.ParTx}(pp, v, k_l, v, sk'_l)$ // after receiving partial transaction tx_l $E := \prod_{l=1}^m tx_l.E \in \mathbb{G}$ if $E \neq \bar{P}' \cdot \bar{P}^{(-1)}$: return \perp $\sigma \leftarrow \mathit{SIG.Sign}(pp, [sk'_l - k_l]_{l=1}^m, E, m)$ $\pi := \mathit{RP.Priv}(pp, \prod_{l=1}^m tx_l.C', v, sk')$ $tx := \mathit{MCT.Combine}(pp, \sigma, \pi, [tx_l]_{l=1}^m, 1, 1)$ return tx</p>

Threshold Key Sharing. We use Shamir secret sharing (SS) scheme [23] with t number of dealers instead of one dealer where t is the threshold. Each dealer separately chooses a secret primary key, and the final key is the summation of all t primary keys. With t dealers, we avoid trust issues of having one dealer and availability issues of having n dealers to generate new keys for each transaction. This protocol is easily modifiable with other secret sharing protocols instead of SS if they support secure multi-dealer sharing where the final key k is recoverable

as a summation of primary keys $[k_j]_{j=1}^t$ s.t. $f(\mathbf{K}) \Rightarrow \sum_{j=1}^t k_j = k$ when \mathbf{K} are the distributed keys among n co-owners.

$\text{SS.Set}(\lambda):$ $\text{return } (p \text{ s.t. prime and } \log_2 p \geq \lambda)$ $\text{SS.ParKey}(pp, i, t, \mathbf{T}, \mathbf{x}, k):$ $\text{if } \mathbf{T} < t \text{ then return } \perp$ $l = 1$ $\text{for } j (j \neq i) \in \mathbf{T} \text{ do}$ $l := l \cdot \frac{x_j}{x_j - x_i}$ $\text{return } k := l \cdot k$	$\text{SS.Dealer}(pp, t, n, K):$ $\mathbf{a} := \{\}, \mathbf{x} := \{\}, \mathbf{k} := \{\}$ $\text{if } K \neq 0 \text{ then } \mathbf{a}_1 := K \text{ else } \mathbf{a}_1 \xleftarrow{\$} \mathbb{Z}_p$ $\text{if } t = 1 \text{ then return } \mathbf{a}_1$ $\text{for } i \in [2, t - 1]: \mathbf{a}_i \xleftarrow{\$} \mathbb{Z}_p$ $\text{for } i \in [1, n]:$ $x_i \xleftarrow{\$} \mathbb{Z}_p, \mathbf{k}_i \leftarrow f(\mathbf{a}, x_i)$ $\text{return } (\mathbf{x}, \mathbf{k})$
---	---

Fig. 1. Threshold key generation when $f(\mathbf{a}, x) = \mathbf{a}_1 + \mathbf{a}_2 x + \dots + \mathbf{a}_{|a|} x^{|a|-1} \pmod p$.

4 Cryptographic Investigation

In this section, we provide the security model of MCT and prove the security of Schnorr-Bulletproofs MCTs and BLS-Bulletproof MCTs (see Sect. 4.4 for proofs).

4.1 Security Model

The completeness of fund transferring denotes that a sender(s) can send a valid transaction to the cash system and pre-transactions to the receiver(s). The receiver(s) can secure the received coins by publishing another transaction with new secret keys. We define the completeness of the whole process of multi-party fund transferring as follows.

Definition 12 (Completeness of Multi-Party Fund Transferring). *MCT is complete if, $[\perp \notin \text{NFT}(pp), \perp \notin \text{TNFT}(pp)]$ is always true when $pp \leftarrow \text{MCT.Set}(\lambda)$ for any λ .*

$$\text{NFT}(pp): n \xleftarrow{\$} \mathcal{N}_{pp}, \mathbf{v} \in \mathcal{V}_{pp}^*, \mathbf{k} \in \mathbb{Z}_p^{n \times |v|}$$

$$\rho \xleftarrow{\$} \mathcal{V}_{pp}^* \text{ s.t. } \sum \mathbf{v} \geq \sum \rho, \mathbf{m} \xleftarrow{\$} \mathcal{N}_{pp}^{|\rho|}, \mathbf{C} \leftarrow \text{MCT.Coin}(\mathbf{v}, [\sum_{j=1}^n (\mathbf{k}_{j,i})^T]_{i=1}^{|v|})$$

$$(tx, \mathbf{ptx}) \leftarrow \text{MCT.Send}(pp, \mathbf{v}, \mathbf{k}, \mathbf{C}, \rho, n, \mathbf{m})$$

$$\mathbf{k} := \{ \sum_{j=1}^n \mathbf{ptx}_{(i,1,j)} \cdot k, \dots, \sum_{j=1}^n \mathbf{ptx}_{(i,m_i,j)} \cdot k \}$$

$$tx \leftarrow [\text{MCT.Receive}(pp, \rho_i, \mathbf{k}, \mathbf{ptx}_i.C, \mathbf{ptx}_i.P, n, \mathbf{m}_i)]_{i \in [1, |\rho|]}$$

$$\text{return MCT.TxVer}(pp, tx) \wedge \prod_{tx' \in tx} \text{MCT.TxVer}(pp, tx')$$

$$\text{TNFT}(pp): t \xleftarrow{\$} \mathcal{N}_{pp}, \mathbf{v} \in \mathcal{V}_{pp}^*, \mathbf{k} \in \mathbb{Z}_p^{t \times |v|}$$

$$\rho \xleftarrow{\$} \mathcal{V}_{pp}^* \text{ s.t. } \sum \mathbf{v} \geq \sum \rho, \mathbf{m} \xleftarrow{\$} \mathcal{N}_{pp}^{|\rho|}, [t_i \xleftarrow{\$} [1, \mathbf{m}_i]_{i=1}^{|\rho|}]$$

$$\mathbf{C} \leftarrow \text{MCT.Coin}(\mathbf{v}, [\sum_{j=1}^t (\mathbf{k}_{j,i})^T]_{i=1}^{|v|})$$

```

    (tx, tptx) ← MCT.Send(pp, v, k, C, ρ, t, m, t)
    for (i ∈ [1, |ρ|]) do
        T ∈§ [1, mi] s.t. |T| = ti // choosing random ti co-owners from all co-owners

        [k'_{(l,j)} ← SS.Parkey(pp, Tl, T, tptx_{(i,l,j)}.x, tptx_{(i,l,j)}.k)]_{(j,l)=(1,1)}^{(t,m_i)}
        txi = MCT.Receive(pp, ρi, {∑_{j=1}^t k_{1,j}, ..., ∑_{j=1}^t k_{m_i,j}}, tptxi.C, t, ti)
    return MCT.TxVer(pp, tx) ∧ ∏_{tx' ∈ tx} MCT.TxVer(pp, tx')
    
```

Confidential transactions are sound if there is no p.p.t adversary who can generate valid transactions with negative coin amounts.

Definition 13 (Computational Soundness). MCT is sound for any λ if $\text{Adv}_{MCT, \mathcal{A}_{RP}}^{CS} = \Pr [\text{Game}_{MCT}^{ZO}(pp) | pp \leftarrow \text{MCT.Set}(\lambda)] \leq \epsilon(\lambda)$

$\text{Game}_{MCT}^{CS}(pp)$: $tx \leftarrow \mathcal{A}_{RP}(pp)$, return $\text{RP.Ver}(pp, tx.C', tx.\pi) \wedge \chi^{\mathcal{E}(\cdot)} \stackrel{?}{\notin} \mathcal{V}_{pp}^{|\pi|}$

We further define Zero Opening Signatures as the property that no p.p.t. adversary can find a signing key for any two openings to values that are different. The name, ZO, is meant to evoke that only commitment pairs, which open to values with zero differences, can have an associated signature key.

Definition 14 (Zero Opening Signatures). MCT satisfies the zero opening signatures property for any p.p.t. adversary \mathcal{A} if

$$\text{Adv}_{MCT}^{ZO} = \Pr [\text{Game}_{MCT}^{ZO}(pp) | pp \leftarrow \text{MCT.Set}(\lambda)] \leq \epsilon(\lambda)$$

$\text{Game}_{MCT}^{ZO}(pp)$: $(x, v, k, v', k') \leftarrow \mathcal{A}(pp)$, $C \leftarrow g^{(k'-k)} h^{(v'-v)}$

return $(v \neq v') \wedge C \stackrel{?}{=} g^x \wedge \text{Sig.Ver}(pp, g^x, g^x, \text{SIG.Sign}(pp, x, g^x))$

Theft Resistance, then, is the property that no p.p.t. adversary can create a valid transaction for a given commitment C , whose opening is (v, k) , using a different key. Since MCT fulfills zero opening, so that no p.p.t. adversary can create a signature when the input coins and output coins have different values, which means that the commitment must be changed if it is to open to the correct v under the different key $k + k'$ (for $k' \neq 0$). The changed commitment can either be guessed from scratch or obtained by the alteration of an existing one. Thus, similarly to what we saw with non-malleability of range proofs, we capture this alteration of commitment with a function $\text{COM.Mal}(C, k')$, which for Pedersen commitments must be of the form $\text{COM.Mal}(C, k') = C \cdot g^{k'}$. The full definition is a bit subtle: it simultaneously considers the cases where the stolen commitment is, either, created from scratch with key $k' \neq 0$ returned to the adversary, or, obtained by malleability through the function $\text{COM.Mal}(C, k')$ whose offset $k' \neq 0$ is returned by the adversary and whose resulting key $k + k'$ is most likely unknown. The adversary does not hint which type of k' is being returned; this is tested by the two predicates connected by the disjunction \vee .

The theft resistance of single-party MCT is not adequate when transactions involves multi-owners. The complete theft-resistance property of N -fund transferring defines that no p.p.t. adversary can create valid transactions for a given commitment C and $\mathbf{k} \setminus \{\mathbf{k}_n\}$ where the blinding factor of C is $\sum_{i=1}^n \mathbf{k}_i$. Complete theft resistance ensures that even an adversary who controls $n - 1$ co-owners can not steal coins.

The complete theft resistance property of T/N -fund transferring implies that less than t co-owners can not create a valid transaction. The worst-case scenario of T/N -fund transferring is $t - 1$ dealers trying to steal the money. In that case, they know $t - 1$ number of primary keys and $t - 1$ of Shamir secret keys of the unknown primary key. Therefore, they can parse $t^2 - 1$ partial keys when they require t^2 total keys.

We define complete theft resistance considering the worst case scenarios as follows,

Definition 15 (Complete Theft-Resistance). *Let $pp \leftarrow \text{MCT.Set}(\lambda)$ for any λ . MCT holds complete theft-resistance property for any p.p.t. adversary \mathcal{A} if,*

$$\text{Adv}_{\text{MCT}}^{\text{CTR}} = \text{Pr} [\text{Game}_{\text{MCT},n}^{\text{CTR}}(pp) \vee \text{Game}_{\text{MCT},t,n}^{\text{CTR}}(pp)] = \epsilon(\lambda)$$

$\text{Game}_{\text{MCT},n}^{\text{CTR}}(pp) :$

$\mathbf{k} \leftarrow \mathbb{Z}_p^n, v \xleftarrow{\$} \mathcal{V}_{pp}$
 $C \leftarrow \text{COM.Cmt}(pp, v, \sum \mathbf{k})$
 $(k', tx) \leftarrow \mathcal{A}(pp, C, v, \mathbf{k} \setminus \{\mathbf{k}_n\})$
 $tx := (C, C', (\pi, E, \sigma))$
 return $(\text{COM.Ver}(pp, C', v, k') \vee$
 $C' \stackrel{?}{=} \text{COM.Mal}(C, k')) \wedge k' \neq 0$
 $\wedge \text{MCT.TxVer}(pp, tx)$

$\text{Game}_{\text{MCT},t,n}^{\text{CTR}}(pp) :$

$\mathbf{k} \leftarrow \text{ParKey}(pp, t, n), v \xleftarrow{\$} \mathcal{V}_{pp}$
 $C \leftarrow \text{COM.Cmt}(pp, v, \sum \mathbf{k})$
 $(k', tx) \leftarrow \mathcal{A}(pp, C, v, \mathbf{k} \setminus \{\mathbf{k}_{t,t}\})$
 $tx := (C, C', (\pi, E, \sigma))$
 return $(\text{COM.Ver}(pp, C', v, k') \vee$
 $C' \stackrel{?}{=} \text{COM.Mal}(C, k')) \wedge k' \neq 0$
 $\wedge \text{MCT.TxVer}(pp, tx)$

// Imitates the behavior of a real threshold fund transferring with random values

$\text{Vote}(t, \mathbf{T}, \mathbf{k}) : \text{return } \{\{\mathbf{k}_{1,T_1}, \dots, \mathbf{k}_{1,T_t}\}, \dots, \{\mathbf{k}_{t,T_1}, \dots, \mathbf{k}_{t,T_t}\}\}$

$\text{KG}(pp, t, n, \mathbf{T}) : \bar{\mathbf{k}} \xleftarrow{\$} \mathbb{Z}_p^t, (x, \mathbf{K}) \leftarrow \forall (j \in \mathbf{T}) : \text{SS.Dealer}(pp, t, n, \bar{\mathbf{k}}_j),$

return $(\bar{\mathbf{k}}, x, \text{Vote}(t, \mathbf{T}, \mathbf{K}))$

$\text{ParKey}(pp, t, n) : \mathbf{T} \subseteq [1, n], (\bar{\mathbf{k}}, x, \mathbf{K}) \leftarrow \text{KG}(pp, t, n, \mathbf{T})$

return $\forall j \in [1, t] : \forall (i \in [1, t]) : \mathbf{k}_{i,j} = \text{SS.ParKey}(pp, \mathbf{T}_i, t, \mathbf{T}, x, \mathbf{K}_{j,T_i})$

Finally, we define the indistinguishability which states that values of the transactions are hidden and can not figure out which input paid which output when there are multiple inputs and outputs.

Definition 16 (Transaction Indistinguishability). *MCT has transaction indistinguishability if, $\text{Adv}_{\text{MCT},\mathcal{A}}^{\text{TI}} = |\text{Pr} [b \stackrel{?}{=} b' | b \xleftarrow{\$} \{0, 1\}, b' \leftarrow \mathcal{A}^{\text{Tx}(\cdot)}(pp)] - \frac{1}{2}| \leq \epsilon(\lambda)$ for a p.p.t adversary \mathcal{A} when $pp \leftarrow \text{MCT.Set}(\lambda)$ for any λ , and with oracle,*

Oracle $\text{Tx}_b(pp)$: $v_0, v_1 \xleftarrow{\$} \mathcal{V}_{pp}^*$ s.t. $|v_0| = |v_1|$,
 $v \xleftarrow{\$} \mathcal{V}_{pp}^*$ s.t. $\sum v = \sum v_b$, $r \xleftarrow{\$} \mathbb{Z}_p^{|v|}$, $r_b \xleftarrow{\$} \mathbb{Z}_p^{|v_b|}$,
 $tx_b \leftarrow \text{MCT.Tx}(pp, v, r, v_b, r_b)$, return tx_b, v_0, v_1

4.2 Security of Compact Schnorr Signatures and BLS Signatures

We build compact MCT with Compact Schnorr Signatures (SIG_{SCH}) [14, 22], Compact BLS Signatures (SIG_{BLS}) [3, 4]. SIG_{SCH} and SIG_{BLS} are explained in Appendix A.

Theorem 2 (Security of Compact Schnorr Signatures). *Compact Schnorr signatures SIG_{SCH} hold completeness and computational EUF-CMA properties if solving DL problem is hard in group \mathbb{G} (prefer [14] for the complete security theorem).*

Theorem 3 (Security of Compact BLS Signatures). *Compact BLS signatures SIG_{BLS} hold completeness and computational EUF-CMA properties if ψ -co-CDH problem is hard in group \mathbb{G}_1 and \mathbb{G} (see [3] for the complete security theorem).*

4.3 Non-malleable, Compact, Multi-party Range Proofs from Bulletproofs

We propose a Logarithmic-sized, Non-Malleable, Non-Interactive, Multi-Party Bulletproof Range Proof scheme with an updated version of improved inner product argument. The improved inner product argument [5, 6] can securely convince a verifier given $(\mathbf{h}, \mathbf{g}, P = \mathbf{h}^l \cdot \mathbf{g}^r, \pi_{\text{IP}})$ that the prover knows (\mathbf{l}, \mathbf{r}) . The fascinating property of IP is it compresses $2|\mathbf{l}|$ elements of (\mathbf{l}, \mathbf{r}) to π_{IP} which only has two \mathbb{Z}_p components and $2 \log_2 |\mathbf{l}| \mathbb{G}$ components (prefer [7] for additional details).

The proposed Non-Malleable MBP only uses strong Fiat Shamir challenges derived from a hash function giving the commitment as an input. The strong Fiat Shamir challenges guarantee non-malleability of RP and IP, making unique challenges for the particular commitment. Here, \mathbf{k}_i is the partial key belong the i th party and v is the coin amount of asset $C_{v, \sum \mathbf{k}}$. MBP allows n parties to generate range proofs for coin amounts in range $[0, 2^l - 1]$ **without** sharing their partial keys with the combiner (prefer [6] for notations. However, we interchange variable symbols g with h and h with g).

RP.Set(λ): return $pp = (\mathbb{G}, \mathbf{g}, \mathbf{h}, g, h, u, p)$ // when a group $\mathbb{G} = \langle g \rangle, \langle h \rangle, \langle u \rangle$ of prime order $p \in \{0, 1\}^\lambda$ and g, h , and u are DL problem hard (NUMS points). \mathbf{g} and \mathbf{h} are generator vectors of \mathbb{G} where $|\mathbf{g}| = |\mathbf{h}| = l$ and $\mathcal{V}_{pp} = [0, 2^l - 1]$.

RP.Prv($pp, C = C_{v, \sum \mathbf{k}}, v, \mathbf{k}, n$):

if $n = \text{empty}$: $n := |\mathbf{k}|$ // The combiner (the dealer) starts the function.

$\mathbf{a}_L \in \{0, 1\}^l$: $\langle \mathbf{a}_L, \mathbf{2}^l \rangle = v$, $\mathbf{a}_R := \mathbf{a}_L - \mathbf{1}^l$

$\mathbf{s}_L, \mathbf{s}_R \stackrel{\$}{\leftarrow} \mathbb{Z}_p^l, (\alpha, \rho) \stackrel{\$}{\leftarrow} \mathbb{Z}_p, A \leftarrow g^\alpha \mathbf{h}^{a_L} \mathbf{g}^{a_R}, S \leftarrow g^\rho \mathbf{h}^{s_L} \mathbf{g}^{s_R}$
 $y \leftarrow H(C, A, S), z \leftarrow H(C, A, S, y)$
 $L(X) := (\mathbf{a}_L - z \cdot \mathbf{1}^l) + \mathbf{s}_L \cdot X, R(X) := \mathbf{y}^l \cdot (\mathbf{a}_R + z \cdot \mathbf{1}^l + \mathbf{s}_R \cdot X) + z^2 \cdot \mathbf{2}^l$
 $t(X) := \langle L(X), R(X) \rangle = t_0 + t_1 \cdot X + t_2 \cdot X^2$
 $\mathbf{gt}_1 = \{\}, \mathbf{gt}_2 = \{\}$
for $i \in [1, n]$ **do**
 // The combiner requests $g^{\tau_{1,i}}, g^{\tau_{2,i}}$ from each co-prover i .
 $\tau_{1,i}, \tau_{2,i} \stackrel{\$}{\leftarrow} \mathbb{Z}_p^2$ // Each co-prover i shares $g^{\tau_{1,i}}, g^{\tau_{2,i}}$ with the combiner j .
 $\mathbf{gt}_1 = \mathbf{gt}_1 \| g^{\tau_{1,i}}, \mathbf{gt}_2 = \mathbf{gt}_2 \| g^{\tau_{2,i}}$ // The combiner adds $g^{\tau_{1,i}}, g^{\tau_{2,i}}$ to an array.
 // The combiner continues the function.
 $T_1 \leftarrow h^{t_1} \prod_{i=1}^n \mathbf{gt}_{1,i}, T_2 \leftarrow h^{t_2} \prod_{i=1}^n \mathbf{gt}_{2,i}$
 $x \leftarrow H(C, A, S, y, z, T_1, T_2), \mathbf{l} := L(x) \in \mathbb{Z}_p^l, \mathbf{r} := R(x) \in \mathbb{Z}_p^l, \hat{t} := \langle \mathbf{l}, \mathbf{r} \rangle$
 $\mathbf{T}_x = \{\}$ // The combiner j shares α, A, S, T_1, T_2 with co-prover $i (i \neq j) \in [1, n]$.
for $i \in [1, n]$ **do**
 if $A \stackrel{?}{\neq} g^\alpha \mathbf{h}^{a_L} \mathbf{g}^{a_R}$: return \perp // Each co-prover verifies the parameters.
 $y := H(C, A, S), z := H(C, A, S, y), x := H(C, A, S, y, z, T_1, T_2)$
 $\tau_{x,i} := \tau_{2,i} \cdot x^2 + \tau_{1,i} \cdot x + z^2 \cdot \mathbf{k}_i$ // Each i shares $\tau_{x,i}$ with the combiner.
 $\tau_x := \sum_{i=1}^n \tau_{x,i}, \mu := \alpha + \rho \cdot x$ // The combiner adds $\tau_{x,i}$ of each i to τ_x .
 $x_{\text{IP}} \leftarrow H(C, A, S, y, z, T_1, T_2, \tau_x, \mu, \hat{t})$
 $\pi_{\text{IP}} := \text{IP.Prove}(pp, \mathbf{h}, \mathbf{g}, u, x_{\text{IP}}, C, \mathbf{h}^l \mathbf{g}^r, \hat{t}, \mathbf{l}, \mathbf{r})$ // IP additionally takes C
 return $\pi := (A, S, T_1, T_2, \tau_x, \mu, \hat{t}, \pi_{\text{IP}})$ // The combiner returns the range proof.

RP.Ver(pp, C, π): // Here $\delta(y, z) = (z - z^2) \cdot \langle \mathbf{1}^l, \mathbf{y}^l \rangle - z^3 \langle \mathbf{1}^l, \mathbf{2}^l \rangle \in \mathbb{Z}_p$
For each $(C, \pi) \in C, \pi := (A, S, T_1, T_2, \tau_x, \mu, \hat{t}, \pi_{\text{IP}}) := \pi$
 $y := H(C, A, S), z := H(C, A, S, y), x := H(C, A, S, y, z, T_1, T_2)$
 $\mathbf{g}' := \mathbf{g}^{(y^{-1})} // \mathbf{g}' = \{\mathbf{g}_1, \mathbf{g}_2^{y^{-1}}, \mathbf{g}_3^{y^{-2}}, \dots, \mathbf{g}_i^{y^{-i+1}}\}$
 if $h^{\hat{t}} g^{\tau_x} \neq C^{z^2} \cdot h^{\delta(y,z)} \cdot T_1^x \cdot T_2^{x^2}$: return *False*
 $P = A \cdot S^x \cdot \mathbf{h}^{-z} \cdot (\mathbf{g}')^{z \cdot \mathbf{y}^l + z^2 \cdot \mathbf{2}^l}$
 $x_{\text{IP}} \leftarrow H(C, A, S, y, z, T_1, T_2, \tau_x, \mu, \hat{t})$
 if $\neg \text{IP.verify}(pp, \mathbf{h}, \mathbf{g}', u, x_{\text{IP}}, C, P g^{-\mu}, \pi_{\text{IP}})$: return *False* // IP takes C
 return *True* (see Appendix B for updated IP)

Theorem 4. *Single-party Bulletproof range proofs [6] explained in [6] have completeness, honest verifier zero-knowledge, and computational knowledge soundness when the discrete log problem is hard in group \mathbb{G} .*

Theorem 5. *Multi-party Bulletproof range proofs (with strong Fiat Shamir challenges) have completeness, honest verifier zero-knowledge, computational knowledge soundness, mon-malleability, and secure against honest-but-curious combiners when the discrete log problem is hard in group \mathbb{G} and single-party Bulletproof range proofs [6] is complete, honest verifier zero-knowledge, and computationally knowledge sound.*

Lemma 1 (Completeness). *MBP is complete if BP is complete.*

Proof. We assign k, τ_1, τ_2 to following MBP values, $k := \sum_{i=1}^n \mathbf{k}_i$, $\tau_1 := \sum_{i=1}^n \tau_{1,i}$, $\tau_2 := \sum_{i=1}^n \tau_{2,i}$. Now we calculate the T_1, T_2, τ_x of MBP as follows,

$$T_1 = h^{t_1} \prod_{i=1}^n g^{\tau_{1,i}} = h^{t_1} g^{\tau_1}, \quad T_2 = h^{t_2} \prod_{i=1}^n g^{\tau_{2,i}} = h^{t_2} g^{\tau_2}$$

$$\tau_x = \sum_{i=1}^n \tau_{x,i} = \sum_{i=1}^n (\tau_{2,i} x^2 + \tau_{1,i} x + z^2 \mathbf{k}_i) = x^2 \sum_{i=1}^n \tau_{2,i} + x \sum_{i=1}^n \tau_{1,i} + z^2 \sum_{i=1}^n x + z^2 \mathbf{k}$$

Here T_1, T_2, τ_x is calculated exactly same as BP [6]. Therefore, we claim that MBP is complete if BP is complete. \square

Lemma 2 (Honest Verifier Zero-Knowledge). *MBP holds honest verifier ZK property if BP holds honest verifier zero-knowledge property.*

Proof. The proof of the above lemma is directly visible as MPB does not change BP protocol [6] except the computation of blinding keys. \square

Lemma 3 (Computational Knowledge Soundness). *MBP holds complete computational knowledge soundness if BP is computationally knowledge sound.*

Proof. Lemma 3 is a directly provable from single-party Bulletproofs [6]. \square

Informally, if BP is knowledge sound, or the extractor always extracts a valid witness given a valid proof, then MBP is also knowledge sound.

Lemma 4 (Non-Malleability). *MBP is computationally non-malleable, in the random-oracle model, if it satisfies computational knowledge soundness.*

Proof. We merely sketch the argument. Since $\text{COM.Mal}(C, k') := C \cdot g^{k'} \neq C$ for $k' \neq 0 \pmod{p}$, the honest-verifier challenge values x, y, z obtained by Fiat-Shamir hashing in the random-oracle model will be random in the new proof, and unrelated to those in the old proof. A careful examination of the protocol reveals that with those new unrelated random challenges, the problem of creating a *related* proof with a shown relation is as hard as that of creating an *unrelated* or fresh proof, which is computationally hard per the computational knowledge soundness property. \square

Lemma 5 (Insider Security). *MBP is secure against honest-but-curious combiners, in the random-oracle model if solving DL problem is hard in \mathbb{G} and BP is zero-knowledge.*

Proof. To prove the insider security, we assume MBP is zero-knowledge. Let there is an adversary \mathcal{A} who breaks the insider security of MBP. Assume \mathcal{A} generates fresh range proofs for $(v, \mathbf{k} \setminus \{\mathbf{k}_n\}, g^{\sum \mathbf{k}})$. Then \mathcal{A} successfully finds τ_x as in $\text{RP.Priv}()$. In other words, \mathcal{A} solves the discrete log problem of g^{k_n} without \mathbf{k}_n . Therefore, we claim that Lemma 5 is true. \square

Finally, we prove that Theorem 5 is true.

4.4 Security Proofs for Compact, Multi-party Confidential Transactions

Now, we prove the security of compact MCT when Schnorr signatures (SIG_{SCH}) [14,22], BLS signatures (SIG_{BLS}) [3,4], and MBP are used.

Theorem 6. *MCT, a compact multi-party confidential transaction protocol is complete, sound, zero-opening, completely theft resistant, and indistinguishable if SIG_{BLS} , SIG_{SCH} , and RP are secure.*

Lemma 6 (Completeness). *MCT is complete if SIG , RP, and CM are complete.*

We point to the MCT construction to prove completeness. Since MCT satisfies verification tests (refer $\text{MCT.TxVer}()$), we claim that MCT is complete.

Lemma 7 (Soundness). *MCT is sound if RP is knowledge sound.*

The validity of Lemma 7 is directly visible from the knowledge soundness of PR.

Lemma 8 (Zero Opening Signatures). *MCT has the zero opening signatures property if COM is binding.*

Proof. We show a reduction breaking the binding property of COM using an adversary who breaks ZO of MCT.

Simulator $\mathcal{S}_{\text{MCT},\mathcal{A}}^{\text{ZO}}$ (pp): $(x, v_1, k_1, v_2, k_2) \leftarrow \mathcal{A}(pp)$, $y = \frac{x-k_2+k_1}{v_2-v_1}$
 $(v, v') \xleftarrow{\$} \mathcal{V}_{pp}$, $k \xleftarrow{\$} \mathbb{Z}_p$, $k' = k - y(v' - v)$, return (v, k, v', k')

Let there is an adversary \mathcal{A} who breaks the zero-opening property of MCT. Then the simulator $\mathcal{S}_{\text{MCT},\mathcal{A}}^{\text{ZO}}$ wins the security game of $\text{Game}_{\text{MCT},\mathcal{S}_{\text{MCT},\mathcal{A}}^{\text{ZO}}}^{\text{BD}}$ using the adversary \mathcal{A} . Since the commitment scheme is binding for any Λ , MCT holds zero-opening property followed by $\text{Adv}_{\text{CM},\mathcal{S}_{\text{MCT},\mathcal{A}}^{\text{ZO}}}^{\text{BD}} = \text{Adv}_{\text{MCT},\mathcal{A}}^{\text{ZO}} = \epsilon(\lambda)$. \square

Lemma 9 (Complete Theft Resistance). *MCT is completely theft resistant if SIG is EUF-CMA and RP is non malleable.*

Simulator $\mathcal{S}_{\text{MCT},n,t,\mathcal{A}}^{\text{CTR}}$ ($pp, P_{\text{SIG}}, C_{\text{RP}}$):

<p>if $t = \text{empty}$: $x = n$ else $x = t * t$ mode $\xleftarrow{\\$} \{\text{SIG}, \text{RP}\}$ if mode = SIG: $\mathbf{k} \xleftarrow{\\$} \mathbb{Z}_p^{x-1}$, $v \xleftarrow{\\$} \mathcal{V}_{pp}$ $C_{\text{SIG}} \leftarrow P^{-1} \cdot \text{CM.Cmt}(pp, v, \sum \mathbf{k})$ if mode = RP: $\mathbf{k} \xleftarrow{\\$} \mathbb{Z}_p^{x-1}$, $v \xleftarrow{\\$} \mathcal{V}_{pp}$ $tx \leftarrow \mathcal{A}(C_{\text{mode}}, v, \mathbf{k})$ $(C, C', (\pi, E, \sigma)) := tx$</p>	<p>if (mode = $\text{SIG} \wedge \text{CM.Ver}(pp, C', v, k')$): return $\sigma := \text{Process}(pp, k', \mathbf{k}, tx)$ if (mode = $\text{RP} \wedge k' \neq 0 \wedge C' = C \cdot g^{k'}$): return π <u>Process</u>(pp, k', \mathbf{k}, tx): if SIG_{BLS}: return $((tx.\sigma) \cdot \text{SIG.Sign}(pp, k' - \Sigma \mathbf{k}, tx.E)^{-1})$ if SIG_{SCH}: return $(tx.\sigma.R, tx.\sigma.s - H(tx.\sigma.R, tx.E) \cdot (k' - \Sigma \mathbf{k}))$</p>
---	---

Proof. We present the following security reduction, where a simulator $\mathcal{S}_{MCT,x,\mathcal{A}}^{CTR}$ uses an adversary \mathcal{A} who breaks theft resistance of MCT to break EUF-CMA of SIG by winning $Game_{SIG,x,\mathcal{S}_{MCT,x,\mathcal{A}}^{CTR}}^{EUF-CMA}$ or NM of RP by winning $Game_{RP,\mathcal{S}_{MCT,x,\mathcal{A}}^{CTR}}^{NM}$. Here, the simulator tries to mimic a challenger for n co-owners or t owners out of n owners. Therefore, when the simulator receives a challenge, it generates parameters for other $n - 1$ partial keys in N -fund transferring or $t * t - 1$ Shamir partial keys for T/N -fund transferring. The simulator randomly chooses a **mode**, to attempt to break either the signature scheme or the range proof scheme. The intuition behind the **mode** is that the simulator does not know the strategy that \mathcal{A} is going to use. Since the simulator guesses the **mode** randomly, there is $1/2$ probability that the **mode** the simulator chose will be suited to exploit a successful break from \mathcal{A} . Therefore, we claim that Lemma 9 is true since $\text{Adv}_{MCT,\mathcal{A}}^{CTR} = 2 \cdot [\text{Adv}_{SIG,\mathcal{S}_{MCT,n,t,\mathcal{A}}^{CTR}}^{EUF-CMA} + \text{Adv}_{RP,\mathcal{S}_{MCT,n,t,\mathcal{A}}^{CTR}}^{NM}] \leq \epsilon(\lambda)$ when RP is non-malleable and SIG is EUF-CMA. \square

Informally, the adversary who has $|\mathbf{sk}| - 1$ number of secret keys out of \mathbf{sk} , tries to forge a transaction tx to generate a new asset with key k' . The adversary has two options as $tx = (C, C \cdot g^{k'}, (\pi_{\sum \mathbf{sk} + k'}, E = g^{k'}, \sigma_{k'}(E)))$ where he can create the signature but not the range proof, or $tx = (C, g^{k'}, (\pi_{k'}, E = g^{k' - \sum \mathbf{sk}}, \sigma_{k' - \sum \mathbf{sk}}(E)))$ where he can create the range proof easily but not the signature. Note that BP [6] allows to generate $\pi_{v,k+k'}$ given $\pi_{v,k}$. Therefore, CT or MCT with [6] are **vulnerable** to theft.

Lemma 10 (Transaction Indistinguishability). *MCT has transaction indistinguishability if COM is hiding and RP is zero-knowledge.*

Proof. Since the coin amount is committed in the commitments and range proof, if COM is computationally/statistically/perfectly hiding and RP is computational/statistically/perfectly zero-knowledge, then MCT has the computational/statistical/perfect transaction indistinguishability property. \square

The foregoing Lemmas 6–10 together prove Theorem 6.

Conclusion. Confidential Transaction Protocol improves the privacy of decentralized cash systems by hiding the transaction amount. The existing Confidential Transaction Protocols support multi-party transactions; however, the size of the transactions is linear to the number of co-owners, and the transactions do not hide the number of co-owners. In this work, we extend Confidential Transaction Protocol to generate compact, private Multi-Party Confidential Transactions resistant to Rogue Key attacks while proving the security of the Schnorr-Bulletproof construction and the BLS-Bulletproof construction.

A Compact Schnorr and BLS Signatures

The protocols of SIG_{SCH} and SIG_{BLS} are explained below.

SIG_{BLS}.Set(λ): return $pp = (\mathbb{G}_1, \mathbb{G}, \mathbb{G}_t, e, g_1, g, p)$, $(\mathbb{G}_1, \mathbb{G}, \mathbb{G}_t)$ of prime order $p \in \{0, 1\}^\lambda$, $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G} = \langle g \rangle$, an efficiently computable non-degenerating pairing $e : \mathbb{G}_1 \times \mathbb{G} \rightarrow \mathbb{G}_t$, and hash function $H : \{0, 1\}^* \rightarrow \mathbb{G}_1$.

SIG.Ver($pp, \mathbf{pk}, \mathbf{m}, \sigma$): **if** $n = \text{empty} : n := 1$
 return $e(\prod \sigma, g_2) \stackrel{?}{=} \prod_{i=1}^{|m|} e(H(m_i), \mathbf{pk}_i)$ // each j creates a partial sig.
 $\sigma_j \leftarrow H(m)^{sk_j}$
 // The combiner(s) aggregates par. sig.

SIG.Sign(pp, \mathbf{sk}, m, n): return $(\prod_{i=1}^n \sigma_i)$

SIG_{SCH}.Set(λ): return $pp = (\mathbb{G}, g, p)$ // \mathbb{G} of prime order $p \in \{0, 1\}^\lambda$, $\mathbb{G} = \langle g \rangle$

SIG.Ver($pp, \mathbf{pk}, \mathbf{m}, \sigma$): // shares R_j with co-signers
 $[(R_i, s_i) := \sigma_i]_{i=1}^{|\sigma|}$ **if** $\neg(\forall(i \in [1, n]) : h_i \stackrel{?}{=} H(R_j))$: return \perp
 return $\forall g^{s_i} \stackrel{?}{=} R_i \cdot \mathbf{pk}_i^{H(R_i, m_i)}$ // each j creates a par.sig.
 $R \leftarrow R_1 \cdot R_2 \cdots R_n$
 $s_j = r_j + H(R, m) \cdot \mathbf{sk}_j$
 // The combiner(s) aggregates partial sig.

SIG.Sign(pp, \mathbf{sk}, m, n): return $(\prod_{i=1}^n R_j, \sum_{i=1}^n s_j)$
if $n = \text{empty} : n := 1$
 // shares $h_j \leftarrow H(R_j)$
 // after receiving $[H(R_i)]_{i=1}^n$

B Improved Inner Product Argument with Strong Fiat Shamir Challenges

// Inner Product Argument - Prove
 // We additionally use C here.
IP.Prove($pp, \mathbf{g}, \mathbf{h}, u, x_{\text{IP}}, C, P, c, \mathbf{a}, \mathbf{b}$)

$P' \leftarrow P \cdot u^{x_{\text{IP}} \cdot c}$
 $(g, h, C, c, P, a, b, \mathbf{l}, \mathbf{r}) := \text{IP.Prove}(pp,$

$\mathbf{h}, \mathbf{g}, C, c, P', \mathbf{a}, \mathbf{b}, \{\}, \{\})$
 // Here $\mathbf{l}, \mathbf{r} \in \mathbb{G}^{\log_2 |a|}$
 return $\pi_{\text{IP}} = (a, b, \mathbf{l}, \mathbf{r})$

// A recursive function
 // $\mathbf{a}_{[n]} = \{\mathbf{a}_1, \dots, \mathbf{a}_{n-1}\}$
 // $\mathbf{a}_{[n]} = \{\mathbf{a}_n, \dots, \mathbf{a}_{|a|}\}$
 // We additionally use C here.
IP.ComputeProof($pp, \mathbf{g}, \mathbf{h}, C, c, P,$
 $\mathbf{a}, \mathbf{b}, \mathbf{l}, \mathbf{r}$)
if $|g| \neq |h| \neq |a| \neq |b|$: return \perp

$n = |g|$
if $n = 1$: return $(\mathbf{g}, \mathbf{h}, C, c, P, \mathbf{a}, \mathbf{b}, \mathbf{l}, \mathbf{r})$

else:

$n' := n/2$
 $c_L \leftarrow \langle \mathbf{a}_{[n']}, \mathbf{b}_{[n']} \rangle \in \mathbb{Z}_p$
 $c_R \leftarrow \langle \mathbf{a}_{[n']}, \mathbf{b}_{[n']} \rangle \in \mathbb{Z}_p$
 $L \leftarrow \langle \mathbf{g}_{[n']}, \mathbf{h}_{[n']} \rangle \in \mathbb{G}$
 $R \leftarrow \langle \mathbf{g}_{[n']}, \mathbf{h}_{[n']} \rangle \in \mathbb{G}$
 // add L, R to \mathbf{l}, \mathbf{r}
 $\mathbf{l} := \mathbf{l} \| L, \mathbf{r} := \mathbf{r} \| R$
 $x \leftarrow H(C, L, R)$
 // element-wise
 $\mathbf{g}' := \mathbf{g}_{[n']}^{x^{-1}} \odot \mathbf{g}_{[n']}^x \in \mathbb{G}^{n'}$
 $\mathbf{h}' := \mathbf{h}_{[n']}^x \odot \mathbf{h}_{[n']}^{x^{-1}} \in \mathbb{G}^{n'}$
 $P' := L^{x^2} P R^{-x^2} \in \mathbb{G}$
 $\mathbf{a}' = \mathbf{a}_{[n']} x + \mathbf{a}_{[n']} x^{-1} \in \mathbb{Z}_p^{n'}$

```

 $b'$  =  $b_{[n']}$  $x^{-1}$  +  $b_{[n':]}$  $x \in \mathbb{Z}_p^{n'}$            if  $\log_2 |\mathbf{g}| \neq \log_2 |\mathbf{h}| \neq |\mathbf{l}| \neq |\mathbf{r}|$ :
( $\mathbf{g}, \mathbf{h}, C, c, P, \mathbf{a}, \mathbf{b}, \mathbf{l}, \mathbf{r}$ ) := ( $\mathbf{g}', \mathbf{h}'$ ,           return  $\perp$ 
   $C, c, P', \mathbf{a}', \mathbf{b}', \mathbf{l}, \mathbf{r}$ )            $n' = |\mathbf{g}|$ 
run recursively IP.ComputeProof(pp), for (L, R)  $\in (\mathbf{l}, \mathbf{r})$ 
   $\mathbf{g}, \mathbf{h}, C, c, P, \mathbf{a}, \mathbf{b}, \mathbf{l}, \mathbf{r}$ )            $n' := n'/2$ 
   $x \leftarrow H(C, L, R)$             $x \leftarrow H(C, L, R)$ 
  // element-wise product  $\odot$ 
  // Inner Product Argument - Verify            $\mathbf{g} := \mathbf{g}_{[n']}^{x^{-1}} \odot \mathbf{g}_{[n']}^x \in \mathbb{G}^{n'}$ 
  // We additionally use C here.            $\mathbf{h} := \mathbf{h}_{[n']}^x \odot \mathbf{h}_{[n']}^{x^{-1}} \in \mathbb{G}^{n'}$ 
  IP.Verify(pp, g, h, u, xIP, C, P, c,  $\pi_{IP}$ )            $P := L^{x^2} P R_i^{-x^2} \in \mathbb{G}$ 
  // Note that  $|\mathbf{g}| = 1 \rightarrow \mathbf{g} = g$  and
   $P \leftarrow P \cdot u^{x_{IP} \cdot c}$             $|\mathbf{h}| = 1 \rightarrow \mathbf{h} = h$ 
  ( $\mathbf{a}, \mathbf{b}, \mathbf{l}, \mathbf{r}$ ) :=  $\pi_{IP}$            return  $P \stackrel{?}{=} g^a h^b u^{x_{IP} \cdot a \cdot b}$ 

```

References

1. Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better — how to make bitcoin a better currency. In: Keromytis, A.D. (ed.) Financial Cryptography and Data Security. LNCS, vol. 7397, pp. 399–414. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_29
2. beam.mw: Scalable confidential cryptocurrency mumblewimble implementation. <https://www.beam.mw/>. Accessed 27 May 2020
3. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 435–464. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_15
4. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_30
5. 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
6. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: efficient range proofs for confidential transactions. In: IEEE SP, May 2018 (2017)
7. Cryptograph, D.: Crate bulletproofs - module bulletproofs::range proof MPC. https://doc-internal.dalek.rs/bulletproofs/range-proof_mpc/index.html. Accessed 21 Nov 2019
8. Fuchsbauer, G., Orrù, M., Seurin, Y.: Aggregate cash systems: a cryptographic investigation of mumblewimble. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 657–689. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_22
9. Horster, P., Michels, M., Petersen, H.: Meta-multisignature schemes based on the discrete logarithm problem. Information Security — the Next Decade. IAICT, pp. 128–142. Springer, Boston, MA (1995). https://doi.org/10.1007/978-0-387-34873-5_11

10. Jedusor, T.E.: Mumblewimble (2016)
11. Langford, S.K.: Weaknesses in some threshold cryptosystems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 74–82. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_6
12. Maxwell, G.: Confidential transactions (2015). [https://people.xiph.org/~\\$greg/confidential_values.txt](https://people.xiph.org/~$greg/confidential_values.txt). Accessed 09 May 2016
13. Maxwell, G., Poelstra, A.: Borromean ring signatures (2015)
14. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to bitcoin, pp. 1–26. *Designs, Codes and Cryptography* (2018)
15. Michels, M., Horster, P.: On the risk of disruption in several multiparty signature schemes. In: Kim, K., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 334–345. Springer, Heidelberg (1996). <https://doi.org/10.1007/BFb0034859>
16. Nakamoto, S.: Bitcoin- a peer-to-peer electronic cash system (2008)
17. 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
18. Poelstra, A.: Mumblewimble (2016). <https://download.wpsoftware.net/bitcoin/wizardry/mumblewimble.pdf>
19. Poelstra, A., Back, A., Friedenbach, M., Maxwell, G., Wuille, P.: Confidential assets. In: Zohar, A., et al. (eds.) *Financial Cryptography and Data Security*. LNCS, vol. 10958, pp. 43–63. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-662-58820-8_4
20. Ristenpart, T., Yilek, S.: The power of proofs-of-possession: securing multiparty signatures against rogue-key attacks. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 228–245. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_13
21. Ruffing, T., Thyagarajan, S., Ronge, V., Schroder, D.: (short paper) burning zero-coins for fun and for profit - a cryptographic denial-of-spending attack on the zero-coin protocol, pp. 116–119 (2018). <https://doi.org/10.1109/CVCBT.2018.00023>
22. Schnorr, C.P.: Efficient signature generation by smart cards. *J. Cryptol.* **4**(3), 161–174 (1991)
23. Shamir, A.: How to share a secret. *Commun. ACM* **22**(11), 612–613 (1979)
24. grin.tech.org: Grin. <https://github.com/mumblewimble>. Accessed 21 May 2020
25. tlu.tarilabs.com: Mumblewimble multiparty bulletproof UTXO. <http://tlu.tarilabs.com/protocols/mumblewimble-mp-bp-utxo/MainReport.html>. Accessed 21 May 2020
26. Wood, G., et al.: Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* **151**, 1–32 (2014)



Simulation Extractable Versions of Groth's zk-SNARK Revisited

Karim Baghery¹, Zaira Pindado²(✉), and Carla Ràfols²

¹ imec-COSIC, KU Leuven, Leuven, Belgium
karim.baghery@kuleuven.be

² Universitat Pompeu Fabra, Barcelona, Spain
{zaira.pindado, carla.rafols}@upf.edu

Abstract. Among various NIZK arguments, zk-SNARKs are the most efficient constructions in terms of proof size and verification which are two critical criteria for large scale applications. Currently, Groth's construction, **Groth16**, from Eurocrypt'16 is the most efficient and widely deployed one. However, it is proven to achieve only knowledge soundness, which does not prevent attacks from the adversaries who have seen simulated proofs. There has been considerable progress in modifying **Groth16** to achieve simulation extractability to guarantee the non-malleability of proofs. We revise the Simulation Extractable (SE) version of **Groth16** proposed by Bowe and Gabizon that has the most efficient prover and **crs** size among the candidates, although it adds Random Oracle (RO) to the original construction. We present a new version which requires 4 parings in the verification, instead of 5. We also get rid of the RO at the cost of a collision resistant hash function and a single new element in the **crs**. Our construction is proven in the *generic group model* and seems to result in the most efficient SE variant of **Groth16** in most dimensions.

Keywords: zk-SNARK · Simulation extractability · Generic group model

1 Introduction

Non-Interactive Zero-Knowledge (NIZK) proof systems are a fundamental family of cryptographic primitives that has appeared recently in a wide range of practical applications. A NIZK proof system allows a party to prove that for a public statement \mathbf{x} , she knows a witness \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathbf{R}$, for some relation \mathbf{R} , without leaking any information about \mathbf{w} and without interaction with the verifier. Due to their impressive advantages, NIZK proof systems are used ubiquitously to build larger cryptographic protocols and systems.

Zero-knowledge Succinct Arguments of Knowledge (zk-SNARKs) are among the most interesting NIZK proof systems in practice, as they allow to generate very short proofs and very efficient verification for NP complete languages [6]. Zk-SNARKs have had a tremendous impact in practice and they have found

numerous applications, including verifiable computation systems [11], privacy-preserving cryptocurrencies [3] and smart contracts [9] and private proof-of-stake protocols [8] are few of known applications that use zk-SNARKs to prove different statements while guaranteeing privacy of the users. Because of their practical importance, particularly in large-scale applications like blockchains, even minimal savings (in proof size or verification cost) are considered to be relevant. In practice, the zk-SNARK is used to prove the correctness of some computations without leaking any information about the secret inputs that are used to complete it. To do so, the computation should be encoded to one of the NP characterizations which currently Quadratic Arithmetic Program (QAP) is the most popular one. The basic idea is that the correctness of all the computations of the circuit is expressed as a divisibility relation among certain polynomials which define the program. Then the characterization can be compiled into a zk-SNARK where the prover gives a proof of knowledge of a witness \mathbf{a} for which the divisibility relation holds for the polynomials which define the QAP combined with the input \mathbf{a} . The succinctness of the argument comes precisely from the fact that the correctness of all the gates is aggregated into just one relation, and that this relation of polynomials is proven in one secret point.

In 2016, Groth [6] introduced the most efficient zk-SNARK in the Generic Group Model (GGM) for QAPs, Groth16, which is still the state-of-the-art. Its proof is 3 group elements and its verification is dominated by 3 pairings. The proof in Groth16 is malleable. Generating non-malleable proofs is a necessary requirement in building various cryptographic schemes, including *universally composable* protocols [8,9], cryptocurrencies (e.g. Zcash) [3], signature-of-knowledge schemes [7], etc.

Therefore, in practice, it is important to have a stronger notion of security, namely, Simulation Extractability (SE). This notion guarantees that a valid witness can be extracted from any adversary producing a proof accepted by the verifier, even after seeing an arbitrary number of simulated queries. For this reason, in Crypto 2017, Groth and Maller [7] proposed a SE zk-SNARK, which is very efficient in terms of proof size but very inefficient in terms of common reference string (crs) size and prover time. Bowe and Gabizon [4] proposed a less efficient construction (5 group elements vs 3) based on Groth16 which adds a Random Oracle (RO) to it but with almost no overhead in crs size or cost for the prover. Recently, Lipmaa [10] proposed several constructions, including the most efficient QAP-based SE zk-SNARK in terms of proof size and with the same verification complexity as [4,7], but less efficient in terms of crs size and prover time compared to [4]. In [1], Atapoor and Baghery used the traditional OR technique to achieve SE in Groth16. Their variant requires 1 pairing less for verification in comparison with previous SE constructions, however it comes with an overhead in proof generation, crs, and even larger overhead in proof size. For a particular instantiation they add $\approx 52,000$ constraints to the underlying QAP instance, which adds fixed overhead to the prover and crs, that can be considerable for mid-size circuits. They show that for a circuit with 10×10^6

Multiplication (Mul) gates, their prover is about 10% slower, but it can be slower for circuits with less than 10×10^6 gates [1].

1.1 Our Contributions

The core of our contribution is revisiting two SE variants of Groth16, presented in [1, 4], to get the best of both constructions. Our focus is mainly on Bove and Gabizon’s variation [4] which has the most efficient prover and the shortest crs among all SE zk-SNARKs [1, 4, 7, 10], while requires a RO. To achieve simulation extractability, their prover replaces all the original computations which depend on some parameter δ given in the crs by some δ' and the prover must give $[\delta']_2$ and a NIZK PoK of DLOG of $[\delta']_2$ w.r.t $[\delta]_2$.

We propose a new SE variant of Groth16 based on Bove and Gabizon’s scheme [4] without ROs. Our variant uses some sophisticated modification of Boneh-Boyen signatures to prove knowledge of the DLOG of δ' and relies only on the collision-resistant properties of a hash function, apart from the GGM. In terms of efficiency, in comparison with [4], our construction requires 1 pairing less in the verification, while retaining all the other properties of their construction. More specifically, the most interesting features of Bove and Gabizon’s scheme are that the crs size and the prover complexity that are almost the same as Groth16 (except for a few exponentiations). Our construction inherits these nice features and avoids using ROs, in the cost of a single new element in the crs which is negligible¹. In comparison with [1], our variant does not have an overhead in proof generation and crs size and it also comes with smaller overhead in proof size.²

Table 1 presents a comparison of our proposed variant of Groth16 with several related constructions for a particular instance of arithmetic circuit satisfiability. Our construction gets rid of the RO in cost of adding one element to the crs.

2 Preliminaries

We let BGgen be a probabilistic polynomial time algorithm which on input 1^λ , where λ is the security parameter, returns the description of an asymmetric bilinear group $\mathbf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{P}_1, \mathcal{P}_2)$, where $\mathbb{G}_1, \mathbb{G}_2$ and \mathbb{G}_T are groups of prime order p , the elements $\mathcal{P}_1, \mathcal{P}_2$ are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively, $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is an efficiently computable, non-degenerate bilinear map, and there is no efficiently computable isomorphism between \mathbb{G}_1 and \mathbb{G}_2 . Elements in \mathbb{G}_γ , are denoted implicitly as $[a]_\gamma = a\mathcal{P}_\gamma$, where $\gamma \in \{1, 2, T\}$ and $\mathcal{P}_T = e(\mathcal{P}_1, \mathcal{P}_2)$. With this notation, $e([a]_1, [b]_2) = [ab]_T$. We extend this notation naturally to vectors and matrices. We denote by $\text{negl}(\lambda)$ an arbitrary negligible function in λ .

¹ In the full version, we show that using a RO we can set $\gamma = 0$ and do not need to add any new element.

² Our changes add only one element to the crs of Groth16 and since the original version is proven to achieve subversion ZK (ZK without trusting a third party) [5], our variant also can be proven to achieve Sub-ZK using the technique proposed in [2].

Table 1. A comparison of our proposed variant of Groth16 along with other SE zk-SNARKs for arithmetic circuit satisfiability with n Mul gates (constraints) and m wires (variables), of which l are public input wires (variables). In the case of crs size and Prover’s computation constants are omitted. In [7], n Mul gates and m wires translate to $2n$ squaring gates and $2m$ wires. In [1], $n' \approx n + 52.000$ and $m' \approx m + 52.000$. \mathbb{G}_1 and \mathbb{G}_2 : group elements, E_i : exponentiation in group \mathbb{G}_i , M_i : multiplication in group \mathbb{G}_i , P : pairings, ROM: Random Oracle Model, CRH: Collision Resistant Hash.

SNARK	Security	Model	crs	Prover	Proof	Verifier
Groth [6]	Knowledge Sound	GGM	$m + 2n - l \mathbb{G}_1$ $n \mathbb{G}_2$	$m + 3n - l E_1$ $n E_2$	$2 \mathbb{G}_1$ $1 \mathbb{G}_2$	$l E_1$ $3 P$
GM [7]	Simulation Extractable	GGM	$2m + 4n \mathbb{G}_1$ $2n \mathbb{G}_2$	$2m + 4n - l E_1$ $2n E_2$	$2 \mathbb{G}_1$ $1 \mathbb{G}_2$	$l E_1$ $5 P$
BG [4]	Simulation Extractable	GGM, ROM	$m + 2n - l \mathbb{G}_1$ $n \mathbb{G}_2$	$m + 3n - l E_1$ $n E_2$	$3 \mathbb{G}_1$ $2 \mathbb{G}_2$	$l E_1$ $5 P$
AB [1]	Simulation Extractable	GGM	$m' + 2n' - l \mathbb{G}_1$ $n' \mathbb{G}_2$	$m' + 3n' - l E_1$ $n' E_2$	$4 \mathbb{G}_1$ $2 \mathbb{G}_2 + 2 \lambda$	$l E_1$ $4 P$
Lipmaa [10]	Simulation Extractable	AGM, Tag-based	$m + 3n - l \mathbb{G}_1$ $n \mathbb{G}_2$	$m + 4n - l E_1$ $n E_2$	$3 \mathbb{G}_1$ $1 \mathbb{G}_2$	$l E_1$ $5 P$
This paper	Simulation Extractable	GGM, CRH	$m + 2n - l \mathbb{G}_1$ $n \mathbb{G}_2$	$m + 3n - l E_1$ $n E_2$	$3 \mathbb{G}_1$ $2 \mathbb{G}_2$	$l E_1$ $4 P$

Security for zk-SNARKs. We use the definitions of NIZK arguments from [6,7]. Our argument is perfectly complete (honest arguments will be accepted with probability 1), perfect zero-knowledge (simulated proofs have the same distribution as honest proofs) and SE (even after seeing v simulated proofs, from any accepting proof output by the adversary it is possible to extract a valid witness).

3 Simulation Extractability Without Random Oracles

In this section, we propose a variation of Groth16 inspired on its Bove and Gabizon [4] SE version. To achieve so, their prover replaces all the computations which depend on δ given in the crs by some δ' of its choice, that it must give as part of the proof, together with a proof of knowledge of the DLOG of δ' w.r.t to δ , which given some element $[Y]_1 = H([A]_1 || [B]_2 || [C]_1 || [\delta']_2)$, consists of $[\pi]_1$ such that $e([Y]_1, [\delta']_2) = e([\pi]_1, [\delta]_2)$. In their analysis, H is an RO and their proof requires 2 pairings for verification. Our contribution is to give an alternative argument of knowledge for the DLOG, with a novel use of Boneh-Boyen signatures along with a proof in the GGM.

Scheme Definition. In Fig. 1, we present our version of Groth16 and we explain in the following how we avoid the use of RO.

Avoiding RO. Our proof uses the collision resistance property of the hash function and the GGM. Very roughly, the new variable γ gives some additional

Setup, $\text{crs} \leftarrow \mathbf{K}(\mathbf{R}, \mathbf{z}_{\mathbf{R}})$: Pick $x, \alpha, \beta, \delta \leftarrow \mathbb{Z}_p^*$, $H \leftarrow \mathcal{H}$ and returns crs , where

$$(\text{crs}_p, \text{crs}_v) := \text{crs} \leftarrow \left(\begin{array}{l} [\alpha, \beta, \delta, \{x^i\}_{i=0}^{n-1}, \{u_j(x)\beta + v_j(x)\alpha + w_j(x)\}_{j=0}^l, \frac{\gamma t(x)}{\delta}] \\ \left[\left\{ \frac{u_j(x)\beta + v_j(x)\alpha + w_j(x)}{\delta} \right\}_{j=l+1}^m, \left\{ \frac{x^i t(x)}{\delta} \right\}_{i=0}^{n-2} \right]_1, \\ [\beta, \delta, \{x^i\}_{i=0}^{n-1}]_2, [\alpha\beta, t(x), \gamma t(x)]_T, H \end{array} \right).$$

Prover, $\pi \leftarrow \mathbf{P}(\mathbf{R}, \mathbf{z}_{\mathbf{R}}, \text{crs}_p, \mathbf{x} = (a_1, \dots, a_l), \mathbf{w} = (a_{l+1}, \dots, a_m))$: with $a_0 = 1$:

1. Select a random element $\zeta \leftarrow \mathbb{Z}_p^*$, and set $[\delta']_2 := \zeta[\delta]_2$
2. Let $A^\dagger(X) \leftarrow \sum_{j=0}^m a_j u_j(X)$, $B^\dagger(X) \leftarrow \sum_{j=0}^m a_j v_j(X)$, $C^\dagger(X) \leftarrow \sum_{j=0}^m a_j w_j(X)$,
3. Set $h(X) = \sum_{i=0}^{n-2} h_i X^i \leftarrow (A^\dagger(X)B^\dagger(X) - C^\dagger(X))/t(X)$,
4. Set $[h(x)t(x)/\delta']_1 \leftarrow (1/\zeta) (\sum_{i=0}^{n-2} h_i [x^i t(x)/\delta]_1)$,
5. Set $r_a \leftarrow_r \mathbb{Z}_p$; Set $[A]_1 \leftarrow \sum_{j=0}^m a_j [u_j(x)]_1 + [\alpha]_1 + r_a [\delta']_1$,
6. Set $r_b \leftarrow_r \mathbb{Z}_p$; Set $[B]_2 \leftarrow \sum_{j=0}^m a_j [v_j(x)]_2 + [\beta]_2 + r_b [\delta']_2$,
7. Set $[C]_1 \leftarrow r_b [A]_1 + r_a \left(\sum_{j=0}^m a_j [w_j(x)]_1 + [\beta]_1 \right) + (1/\zeta) \sum_{j=l+1}^m a_j ([u_j(x)\beta + v_j(x)\alpha + w_j(x)]/\delta)_1 + [h(x)t(x)/\delta']_1$,
8. Sets $m = H([A]_1 \parallel [B]_2 \parallel [C]_1 \parallel [\delta']_2)$,
9. Compute $[D]_1 = \frac{m}{\zeta+m} [\frac{t(x)}{\delta}]_1 + \frac{1}{\zeta+m} [\frac{\gamma t(x)}{\delta}]_1 = [\frac{(m+\gamma)t(x)}{\delta'+m\delta}]_1$
10. Return $\pi := ([A, C, D]_1, [B, \delta']_2)$.

Verifier, $\{1, 0\} \leftarrow \mathbf{V}(\mathbf{R}, \mathbf{z}_{\mathbf{R}}, \text{crs}_v, \mathbf{x} = (a_1, \dots, a_l), \pi = ([A, C, D]_1, [B, \delta']_2))$:

assuming $a_0 = 1$, and setting $m = H([A]_1 \parallel [B]_2 \parallel [C]_1 \parallel [\delta']_2)$ check if

1. $[A]_1 [B]_2 = [C]_1 [\delta']_2 + \left(\sum_{j=0}^l a_j [u_j(x)\beta + v_j(x)\alpha + w_j(x)]_1 \right) [1]_2 + [\alpha\beta]_T$
2. $[D]_1 [\delta' + \delta m]_2 = m [t(x)]_T + [\gamma t(x)]_T$

and return 1 if both checks pass, otherwise return 0.

Simulator, $\pi \leftarrow \mathbf{Sim}(\mathbf{R}, \mathbf{z}_{\mathbf{R}}, \text{crs}_v, \mathbf{x} = (a_1, \dots, a_l), \mathbf{ts})$: Given the simulation trapdoors $\mathbf{ts} := (x, \alpha, \beta, \delta)$ act as follows,

1. Choose random $\zeta \leftarrow_r \mathbb{Z}_p^*$ and set $\delta' := \zeta\delta$
2. Choose $A, B \leftarrow_r \mathbb{Z}_p$ and let $C = (A \cdot B - \sum_{j=0}^l a_j (u_j(x)\beta + v_j(x)\alpha + w_j(x) - \alpha\beta)/\delta')$
3. Let $m = H([A]_1 \parallel [B]_2 \parallel [C]_1 \parallel [\delta']_2)$
4. $[D]_1 = \frac{m}{\zeta+m} [\frac{t(x)}{\delta}]_1 + \frac{1}{\zeta+m} [\frac{\gamma t(x)}{\delta}]_1 = [\frac{(m+\gamma)t(x)}{\delta'+m\delta}]_1$

Fig. 1. The proposed variation of Groth16 for \mathbf{R} . \mathcal{H} is a family of collision resistant hash functions that map to \mathbb{Z}_p^* . The elements $[\alpha\beta, t(x), \gamma t(x)]_T$ are redundant and can in fact be computed from the rest of the elements in the crs . Differences with Groth16 are highlighted. Alternatively, one can describe Groth16 as corresponding to $\zeta = 1, \gamma = 0$ and where the proof consists only of $[A, C]_1, [B]_2$.

guarantees because to compute $t(x) \frac{(\gamma+m)}{(\delta'+\delta m)}$ from D_j such that $m_j \neq m$, it is necessary to know both $\frac{1}{(\delta'+\delta m)}$ and $\frac{\gamma}{(\delta'+\delta m)}$, but this is only possible when $\delta' + \delta m = \zeta\delta$.

Security. We prove security of new scheme (Fig. 1) in Theorem 1.

Theorem 1 (Completeness, ZK, SE). *The variation of Groth16 described in Fig. 1, guarantees (1) perfect completeness, 2) perfect zero-knowledge and 3) simulation-extractability in the asymmetric Generic Group Model.*

Proof. Perfect completeness and perfect zero-knowledge are obvious and the proof is omitted. Knowledge extractability is proven by reduction (in the GGM) to the knowledge soundness of Groth16. The reduction works in two steps (similarly to [4], although the proof of each of these steps is different):

Step 1. Extraction of the DLOG of δ' .

Step 2. Reduction to the Knowledge Soundness of Groth16.

Proof of Step 1) Suppose \mathcal{A} has made queries $\mathbf{x}_1, \dots, \mathbf{x}_v$ to $\text{Sim}(\mathbf{ts}, \cdot)$, and received answers $\{\pi_j = ([A_j]_1, [B_j]_2, [C_j]_1, [D_j]_1, [\delta'_j]_2)\}_{j=1}^v$. Let Q' be the union of elements in the crs together with those from the replies of $\text{Sim}(\mathbf{ts}, \cdot)$; namely,

$$Q' := \left(\begin{array}{l} [\alpha, \beta, \delta, \{x^i\}_{i=0}^{n-1}, \\ \{u_j(x)\beta + v_j(x)\alpha + w_j(x)\}_{j=0}^l, \\ \{u_j(x)\beta + v_j(x)\alpha + w_j(x) \\ \delta\}_{j=t+1}^m, \\ \{x^i t(x)/\delta\}_{i=0}^{n-2}, \gamma t(x)/\delta]_1, [\beta, \delta, \{x^i\}_{i=0}^{n-1}]_2 \end{array} \right) \cup \left(\begin{array}{l} \left\{ \left[\begin{array}{l} A_j, C_j := \frac{A_j B_j - ic_j - \alpha\beta}{\delta_j}, \\ D_j := \frac{t(x)(\gamma + m_j)}{\delta_j + m_j \delta} \end{array} \right]_1 \right\} \\ [B_j, \delta'_j]_2, m_j \}_{j=1}^v \end{array} \right)$$

where $ic_j = \sum_{i=0}^l \alpha_i^j (u_i(x)\beta + v_i(x)\alpha + w_i(x))$, $\mathbf{x}_j = (a_1^j, \dots, a_l^j)$, and $m_j \in \mathbb{Z}_p$ the message that simulator receives from the hash function for each A_j, B_j, C_j, δ_j .

We assume \mathcal{A} has produced the elements (A, B, C, D, δ') such that $A \cdot B \equiv C \cdot \delta' + \left(\sum_{j=0}^l a_j(u_j(x)\beta + v_j(x)\alpha + w_j(x))\right) + \alpha\beta$, for $m := H([A]_1 \parallel [B]_2 \parallel [C]_1 \parallel [\delta']_2)$, $D(\delta' + \delta m) = t(x)(m + \gamma)$. Let Q'_1 be the elements of Q' in \mathbb{G}_1 and Q'_2 the elements in \mathbb{G}_2 . Since the adversary is generic it has constructed these elements as a linear combination of the elements in Q' which are in the relevant group (i.e. element of Q'_1 in \mathbb{G}_1 for A, C, D and Q'_2 for B, δ') and we can extract the coefficients of this linear combination.

First, we prove that \mathcal{A} has knowledge of the DLOG of δ' w.r.t. δ . From the second verification equation we know that $D = t(x) \frac{\gamma + m}{\delta' + m\delta}$. On the other hand, from adversary \mathcal{A} we can recover a vector \mathbf{k}_D with the coefficients that it has used to construct D , that is, $D = \sum_{q \in Q'_1} k_{D,q} q$. Equating these two expressions,

$$t(x)(m + \gamma) = \left(\sum_{q \in Q'_1} k_{D,q} q\right)(\delta' + m\delta), \tag{1}$$

where $\delta' = \sum_{q \in Q'_2} k_{\delta',q} q$ for another vector of coefficients $\mathbf{k}_{\delta'}$. The terms which include γ in both sides of the equation must be the same.

On the other hand, by assumption, in the asymmetric GGM, the term δ' is constructed as a linear combination of elements in Q'_2 and therefore $\delta' + \delta m$ is independent of γ . Then, keeping only the terms with γ in Eq. (1), we obtain

$$t(x)\gamma = k_D \frac{\gamma t(x)}{\delta} (\delta' + m\delta) + \sum_{j=1}^v k_{D,j} \frac{\gamma t(x)}{\delta_j + m_j \delta} (\delta' + m\delta). \tag{2}$$

Dividing both sides of the equation by $t(x)\gamma$ and defining $k_{D,0} = k_D$, $\delta_0 = \delta'$, $m_0 = 0$, we obtain the following equivalent equation:

$$1 = \left(\sum_{j=0}^v k_{D,j} \frac{1}{\delta_j + m_j \delta} \right) (\delta' + m\delta) = \sum_{j=0}^v k_{D,j} \frac{\prod_{i=0, i \neq j}^v (\delta_i + m_i \delta)}{\prod_{i=0}^v (\delta_i + m_i \delta)} (\delta' + m\delta)$$

$$\Leftrightarrow \prod_{i=0}^v (\delta_i + m_i \delta) = (\delta' + m\delta) \left(\sum_{j=0}^v k_{D,j} \prod_{i=0, i \neq j}^v (\delta_i + m_i \delta) \right). \quad (3)$$

It follows that the term $\delta' + m\delta$ must divide the left side of Eq. (3). Therefore, there exists some index j^* and $k \in \mathbb{Z}_p$ such that $\delta' + m\delta = k(\delta_{j^*} + m_{j^*}\delta)$. Now, dividing Eq. (3) by $(\delta_{j^*} + m_{j^*}\delta)$, we come to the following expression:

$$0 = (1 - k \cdot k_{D,j^*}) \prod_{i=0, i \neq j^*}^v (\delta_i + m_i \delta) - k \cdot \sum_{j=0, j \neq j^*}^v k_{D,j} \prod_{i=0, i \neq j}^v (\delta_i + m_i \delta).$$

Since all summands are linearly independent polynomials, $k = k_{D,j^*}^{-1}$, and $k_{D,j} = 0$ if $j \neq j^*$. We distinguish two cases: (1) $\delta' + \delta m = k\delta$, in which case we can extract the DLOG of δ' as $k - m$ as wanted, or (2) $\delta' + \delta m = k(\delta_{j^*} + m_{j^*}\delta)$, in which case, from Eq. (1) and putting everything together, we have that:

$$t(x)(m + \gamma) = k_{D,j^*} \frac{(\gamma + m_{j^*})}{(\delta_{j^*} + m_{j^*}\delta)} (\delta' + m\delta) = k_{D,j^*} k^{-1} (\gamma + m_{j^*}) t(x) = (\gamma + m_{j^*}) t(x).$$

This implies that $m_{j^*}^* = m$ is a collision of H .

Proof of Step 2) We show that the elements A, B, C do not use the elements of the simulated proofs, say $V := \{[A_j]_1, [B_j]_2, [C_j]_1, [D_j]_1, [\delta_j]_2\}_{j=1}^v$, and then, with the knowledge of ζ such that $\delta' = \zeta\delta$, we can reduce our proof to the knowledge soundness proof of Groth16 [6], since $[A]_1, [B]_2, [C\zeta]_1$ is a valid proof of Groth16. For this, we need to argue that A, B, C cannot have been constructed from any of the elements of the queries. To prove that A, B, C are not constructed from the elements $[A_j]_1, [B_j]_2, [C_j]_1, [\delta_j]_2$, we follow the exact same reasoning as Bowe and Gabizon [4] in the GGM and we omit the details. Next, we prove that to construct A, C the prover cannot have used any of the D_j terms, which are the new elements in our proof.

Assume A has been generated from some D_j , so the term $\frac{t(x)(m_j + \gamma)}{\delta_j + m_j \delta}$ appears in the expression of A generated from Q'_1 with the corresponding coefficient different from 0. Observe that the verification equation contains the term $\alpha\beta$ that cannot be manipulated because it is fixed in the crs , and it should be produced by the term AB because $\beta \in Q'_2$ and β is independent of $\delta' = \zeta\delta$. In that case, the product AB would contain a term $\frac{t(x)(m_j + \gamma)}{\delta_j + m_j \delta} \beta$, but this cannot be cancelled out by any of the other terms in the equation. Indeed, this term cannot appear in $\alpha\beta$, or in the sum of public values of a_i . Thus, the only possibility is that it appears in $C\delta'$. However, since β is independent of δ' , it should appear in C , but $\frac{t(x)(m_j + \gamma)}{\delta_j + m_j \delta} \beta$ cannot be computed from elements in Q'_1 .

If D_j appears in C , then the term $C\delta'$ includes $\frac{t(x)(m_j+\gamma)}{\delta_j+m_j\delta}\delta'$. Neither the term $\alpha\beta$ nor the sum of public values can include it, so it can only appear in AB . Since $\delta' \in Q'_2$, then A would contain D_j , which we ruled out previously. \square

4 Conclusion

Over the last few years, various zk-SNARKs have been proposed that achieve simulation extractability [1, 4, 7, 10], which is a requirement for zk-SNARKs to generate non-malleable proofs. In this paper, we revised the SE variation of Groth16 proposed in [4] and presented a new one. Our construction requires 4 pairings in verification, instead of 5 in [4], and also avoids ROs in exchange for using a collision resistant hash function. It has a more efficient prover, crs size, and proof size in comparison with [1], that has also 4 pairings in verification.

Acknowledgements. The research leading to this article was partially supported by Project RTI2018-102112-B-I00 (AEI/FEDER, UE), Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0085, and by Cyber Security Research Flanders with reference number VR20192203.

References

1. Atapoor, S., Bagheri, K.: Simulation extractability in Groth's zk-SNARK. In: Pérez-Solà, C., Navarro-Arribas, G., Biryukov, A., Garcia-Alfaro, J. (eds.) DPM/CBT -2019. LNCS, vol. 11737, pp. 336–354. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31500-9_22
2. Bagheri, K.: Subversion-resistant simulation (knowledge) sound NIZKs. In: Albrecht, M. (ed.) IMACC 2019. LNCS, vol. 11929, pp. 42–63. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35199-1_3
3. 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
4. Bove, S., Gabizon, A.: Making groth's zk-SNARK simulation extractable in the random oracle model. Cryptology ePrint Archive, Report 2018/187 (2018). <https://eprint.iacr.org/2018/187>
5. Fuchsbauer, G.: Subversion-zero-knowledge SNARKs. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. LNCS, vol. 10769, pp. 315–347. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76578-5_11
6. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
7. Groth, J., Maller, M.: Snarky signatures: minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 581–612. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_20
8. Kerber, T., Kiayias, A., Kohlweiss, M., Zikas, V.: Ouroboros cryptsinous: privacy-preserving proof-of-stake. In: 2019 IEEE Symposium on Security and Privacy, pp. 157–174. IEEE Computer Society Press (2019)

9. Kosba, A.E., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE Symposium on Security and Privacy, pp. 839–858. IEEE Computer Society Press, May 2016
10. Lipmaa, H.: Simulation-extractable SNARKs revisited. Cryptology ePrint Archive, Report 2019/612 (2019). <http://eprint.iacr.org/2019/612>
11. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, pp. 238–252. IEEE Computer Society Press, May 2013



Efficient Composable Oblivious Transfer from CDH in the Global Random Oracle Model

Bernardo David^{1(✉)} and Rafael Dowsley²

¹ IT University of Copenhagen, Copenhagen, Denmark
beda@itu.dk

² Monash University, Melbourne, Australia

Abstract. Oblivious Transfer (OT) is a fundamental cryptographic protocol that finds a number of applications, in particular, as an essential building block for two-party and multi-party computation. We construct the first universally composable (UC) protocol for oblivious transfer secure against active static adversaries based on the Computational Diffie-Hellman (CDH) assumption. Our protocol is proven secure in the observable Global Random Oracle model. We start by constructing a protocol that realizes an OT functionality with a selective failure issue, but shown to be sufficient to instantiate efficient OT extension protocols. In terms of complexity, this protocol only requires the computation of 6 modular exponentiations and the communication of 5 group elements, five binary strings of security parameter length, and two binary strings of message length. Finally, we lift this weak construction to obtain a protocol that realizes the standard OT functionality (without any selective failures) at an additional cost of computing 9 modular exponentiations and communicating 4 group elements, four binary strings of security parameter length and two binary strings of message length. As an intermediate step before constructing our CDH based protocols, we design generic OT protocols from any OW-CPA secure public-key encryption scheme with certain properties, which could potentially be instantiated from more assumptions other than CDH.

1 Introduction

Oblivious transfer (OT) [26, 37] is a fundamental cryptographic primitive that serves as a building block for a number of interesting applications, such as secure two-party and multi-party computation. In this work, we mainly focus on 1-out-of-2 string oblivious transfer, which is a two-party primitive. In this flavor of OT, the sender Alice inputs two strings m_0 and m_1 , and the receiver Bob inputs a choice bit c , obtaining m_c as the output. Bob must not be able to learn

B. David—This work was supported by a grant from Concordium Foundation and by Independent Research Fund Denmark grants number 9040-00399B (TrA²C) and number 9131-00075B (PUMA).

m_{1-c} , while Alice must not learn c . Since oblivious transfer is normally used within other protocols as a primitive, it is desirable to ensure that its security is guaranteed even under arbitrary composition.

The Universal Composability (UC) framework [7] is the most widely used methodology for analyzing protocol security under arbitrary composition. OT protocols UC-secure against static malicious adversaries can be designed under several computational assumptions, such as: Decisional Diffie-Hellman (DDH) [28, 36], strong RSA [28], Quadratic Residuosity (QR) [36], Decisional Linear (DLIN) [16, 32], Decisional Composite Residuosity (DCR) [13, 32], McEliece Assumptions [19], low noise Learning Parity with Noise (LPN) [18] and Learning with Errors (LWE) [36]. Furthermore, there exist constructions based on simple generic primitives such as enhanced trapdoor functions [11] and public-key encryption plus semi-honest stand alone oblivious transfer [31], which mostly do not achieve the same efficiency as the constructions that leverage properties of specific computational assumptions.

It is a well-known fact that UC-secure OT protocols require a setup assumption [9]. Coincidentally, most of the UC-secure OT protocols (including the aforementioned ones) are based in the Common Reference String (CRS) model, where the parties are assumed to have access to a string randomly sampled from a given distribution before execution starts. While this setup assumption allows for the construction of efficient UC-secure OT protocols under a number of assumptions, questions have been raised about its practicality [10, 14], since a local CRS is not readily available for a real world implementation of a protocol. Notice that OT can be UC-realized under a number of alternative setup assumptions, such as the public-key infrastructure model [15], the random oracle model (ROM) [3, 5], noisy channels [24], tamper-proof hardware [23, 25, 33]. However, these models still require each instance of the protocol to access a local instance of the setup assumption. Informally, it means that each instance of the protocol uses an instance of the ideal functionality representing the setup assumption that is independent from all other instances and accessible only to the parties participating in the protocol execution but not to the environment.

Assuming that each protocol instance has local access to an independent setup in order to obtain secure composition is far from optimal and results in several issues that have been pointed out in previous works [4, 8, 10]. In particular, assuming the existence of independent random oracles (RO) for each protocol instance contradicts the common practice of replacing a random oracle by a standardized hash function, which is freely accessible and used by everybody. Such issues were first analyzed and addressed by Canetti *et al.* [8], who proposed the “Generalized UC model”, where it is assumed that the instance of the trusted setup is globally available (and therefore also accessible by the environment) and used by all protocol instances. This formalism was subsequently extended to the random oracle setting by Canetti *et al.* [10], who define a global random oracle model, where a single instance of the random oracle \mathcal{F}_{gRO} is directly accessible by all parties, the adversary and the environment. Such a model precludes the use of proof techniques that require the simulator to “program” the random oracle’s

answers to a given query, which are usually employed in random oracle based constructions. UC protocols based on a local programmable CRS also suffer from issues similar to those of local programmable ROs [10], and formally the security guarantees for protocols based on local setups (e.g. local CRS or programmable RO) only hold if a new fresh setup is available for each individual instance of the protocol, which is unrealistic. It is not known how to generate even a single CRS without heuristics, let alone a fresh one for each execution. Quoting Canetti *et al.* [10] on the strength of the global random oracle model: “This model provides significantly stronger composable security guarantees than the traditional random oracle model of Bellare and Rogaway [3] or even the common reference string model”. Note that more than one trusted setup instance can be available (in our construction we use 3 instances of global RO), but they should be globally available and not local for a protocol instance. Surprisingly, Canetti *et al.* [10] showed that using \mathcal{F}_{gRO} as a setup assumption it is possible to construct universally composable DLOG based commitments and DDH based two-party computation and non-interactive secure computation secure against static malicious adversaries. Recently, new results in the global ROM were proven assuming certain relaxations of the model [6]. However, no efficient oblivious transfer protocol in the global random oracle model has been proposed so far.

1.1 Our Contributions

We first propose a generic protocol for universally composable oblivious transfer secure against active static adversaries in the global random oracle model of [10]. The central building block of this construction is a One-Way Chosen Plaintext Attack (OW-CPA) secure public-key encryption (PKE) scheme with a number of properties. We show that such a scheme can be efficiently instantiated under the Computational Diffie Hellman (CDH) assumption. Our results can be summarized as follows:

- The *first* UC-secure OT protocol based on the CDH assumption.¹
- The first UC-secure OT protocol in the Global Random Oracle model [10] that achieves efficiency for single executions (without OT extension) comparable to the most efficient previous work [36], which requires a programmable CRS.²

In order to obtain a protocol based on an assumption as weak as CDH, we introduce novel simulation techniques for extracting choice bits and messages in the simulation without resorting to programming the random oracle, which is not possible in the global random oracle model of Canetti *et al.* [10]. Notice that previous works required stronger computational assumptions (e.g. DDH [5, 36]) even though they relied on stronger local setup assumptions (e.g. CRS [36] and programmable random oracles [5]). Hence, in comparison to such previous works,

¹ Döttling *et al.* [22] proposed an independent UC secure OT protocol in the CRS model with other techniques that yield CDH instantiations.

² The DDH based NISC of [10] is orders of magnitude less efficient than our approach and the protocol [12] has been introduced recently as independent work.

our results improve on both the computational and setup assumptions required for UC-secure OT.

In terms of efficiency, our protocols compare favorably to previous works based on stronger assumptions. In the setting where one wishes to execute a large number of OTs through OT extension, the costs of each seed OT with our CDH based protocol are only the computation of 6 modular exponentiations and the communication of 5 group elements, 5 binary strings of security parameter length, and 2 binary strings of message length. In the setting where few OTs are needed, our CDH based protocol requires 15 modular exponentiations and the communication of 9 group elements, 9 binary strings of security parameter length, and 4 binary strings of message length. We remark that, in contrast to previous works based on local setup assumptions, our protocols can be readily implemented while retaining their security properties by substituting the global random oracle by an extensively tested cryptographic hash function (*e.g.* SHA3).

As an intermediate step towards our CDH based construction, we first design a generic protocol based on a public-key encryption scheme with certain properties. We start by constructing a generic protocol that realizes an OT functionality that captures a selective failure issue, which is nevertheless sufficient for instantiating efficient OT extension protocols as shown in [21]. Interestingly, our protocol achieves high efficiency, requiring only one key generation operation, two encryption operations and one decryption operation, apart from a few calls to the random oracle. In terms of communication, our protocol only requires the transfer of one public-key, two ciphertexts, five binary strings of security parameter length, and two binary strings of message length. Later on, we obtain a generic protocol that realizes the standard OT functionality (without any selective failure) by augmenting our original protocol with four encryptions, one decryption, two ciphertexts, two binary strings of security parameter length and two binary strings of message length. If hundreds of OTs are needed, our OT with selective failures represents a new option of base OT for use with OT extension schemes. If only tens of OTs are needed, our OT without selective failures is a good option for usage. Besides yielding a CDH based instantiation, these generic protocols can be potentially instantiated under other assumptions, paving the way to post-quantum secure constructions of UC-secure OT under lattice and coding based assumptions.

1.2 Related Works

The global random oracle model has been established by Canetti *et al.* in [10], where they also build UC-secure commitments, two-party computation and non-interactive secure computation (NISC) secure against static malicious adversaries. In their construction of NISC in the global ROM, they state that a natural way to construct such a protocol would be to instantiate existing approaches based on 2-round OT with a global ROM version of the originally CRS based UC-secure OT protocol by Peikert *et al.* [36]. However, they observe that there are significant challenges in obtaining such a global ROM version of the protocol by Peikert *et al.*, and instead construct a one-side simulatable OT protocol that

is only UC-secure against a malicious receiver. Their solution is *not generic* but intrinsically based on DDH via non-black-box use of the OT protocol of [36], only implying 2-round UC OT based on DDH, and with communication/computation costs several orders of magnitude higher than ours. On the other hand, ours is the first UC OT in the GRO built in a black-box way from a generic primitive (a PKE that we define), yielding the first UC OT based on CDH (a weaker assumption) while achieving much lower computation/communication costs. Even though the global ROM was recently revisited in [6], allowing for relaxations such as programming the random oracle in specific situations, no new results related to oblivious transfer were proposed in this relaxed model.

The idea of constructing OT using two public-keys—the “pre-computed” one and the “randomized” one dates back to early days of OT development [2, 26]. Naor and Pinkas [35] presented an improved stand alone CDH-based protocol in the (local) random oracle model under the same approach that is proven secure in the half-simulation paradigm. A recent result by Friolo *et al.* [27] shows how to construct 4 round fully simulatable OT from key agreement protocols with certain properties without requiring setup assumptions, which yields a protocol based on CDH. However, their results fundamentally fall short of UC security (since UC-secure OT protocols necessarily require a setup assumption [9]) and cannot be easily adapted to this setting.

We remark that the “Simplest OT” protocol [14] and the protocol by Hauck and Loss [30] have been found to suffer from a number of issues [5, 29] and are not UC secure. The CDH based protocol of [21] only realizes an OT functionality with a selective failure (as our first simple construction) and it is unclear how to use it to realize the standard OT functionality (without selective failure). The UC OT protocol of [1] can also be instantiated from a similar generic public key encryption scheme, for which a CDH instantiation is presented (among other assumptions). However, in order to prove the security of the construction of [1], it is also necessary to assume that the public key encryption scheme has circular security, which is an ad-hoc assumption not proven under CDH.

Independent and Concurrent Works: Döttling *et al.* [22] proposed a generic round optimal UC-secure OT protocol in the CRS model that can be instantiated from CDH. However, even though their protocol solves the important problem of achieving round optimality, it has computational and communication complexities orders of magnitude higher than our protocol, making it impractical. These overheads are intrinsic to the use of generic zero-knowledge proofs and garbled circuits in their construction. Canetti *et al.* [12] introduced a CDH based OT protocol that is UC-secure in the Global Random Oracle model. Similarly to our initial result, they focus on obtaining OT with selective failures in order to achieve better efficiency when using their protocol as basis for OT extension. However, differently from our final result, they do not show how to eliminate selective failures in their protocol without using OT extension.

1.3 Our Techniques

At a high level, we start by building a simple generic protocol that realizes a weak version of the OT functionality, which allows for a selective failure attack. Starting from this weak flavor of OT is useful because it allows us to showcase our techniques more clearly while still being useful for performing OT extension, which results in an unlimited number of standard OTs (without selective failures) at very high efficiency. We then lift our protocol with selective failures to a generic protocol that realizes the standard OT functionality by leveraging subtleties of the first, simpler, construction. The central building block for both protocols is a public-key encryption (PKE) scheme satisfying a number of properties, which we construct based on the CDH assumption departing from the ElGamal cryptosystem. In order to provide some intuition on the design of our schemes, we informally describe properties we require from our PKE scheme and discuss how they are used to build our protocols:

- *Property 1 (informal)*: Let the public-key space \mathcal{PK} form a group with operation denoted by “ \star ”. Then, for the public-keys (pk_0, pk_1) , such that $pk_0 \star pk_1 = q$, where q is chosen uniformly at random from \mathcal{PK} , one cannot decrypt both ciphertexts encrypted using pk_0 and pk_1 , respectively. In particular, when a public/secret-key pair (pk_c, sk_c) is generated, the above relationship guarantees that pk_{1-c} that is chosen to satisfy the constraint $pk_0 \star pk_1 = q$ is “substantially random”, so that learning the messages encrypted with pk_{1-c} is hard.
- *Property 2 (informal)*: pk obtained using the key generation algorithm is indistinguishable from a random element of \mathcal{PK} . Note that we assume in this work that not all the elements of \mathcal{PK} may represent valid public-keys.
- *Property 3 (informal)*: The PKE scheme must be “committing”, meaning that it must be impossible to generate two pairs of randomness and plaintext messages (r_0, m_0) and (r_1, m_1) with $m_0 \neq m_1$ such that encrypting m_0 with randomness r_0 under a uniformly random public-key pk yields the same ciphertext as encrypting m_1 with randomness r_1 under the same public-key.
- *Property 4 (informal)*: Property 3 only holds for key pairs generated according to the key generation algorithm or picked at random, but not for arbitrary key pairs, which could be crafted to be “non-committing”. Intuitively, this property says that encrypting a message under such an arbitrary “non-committing” public key will also cause some message bits to be lost, which will come in handy in the security proof.
- *Property 5 (informal)*: The PKE scheme has a witness-recovering decryption algorithm that outputs the randomness used to generate the decrypted ciphertext along with the plaintext message.

A Toy Example: Consider a very simple protocol where the receiver generates a key pair (pk_c, sk_c) , queries a global RO with a random seed value s to obtain q , computes pk_{1-c} such that $pk_0 \star pk_1 = q$, and sends pk_0 and s to the sender. The latter recomputes pk_1 from pk_0 and s with the help of the RO and uses the public-keys to encrypt random seeds. The sender then uses these seeds to generate

one-time pads (using the global RO) that she uses to encrypt her messages, sending both the PKE ciphertexts containing the seeds and the one-time pad encryptions of the actual messages to the receiver. The receiver can retrieve the seed encrypted under pk_c (since he has sk_c), compute the one-time pad with the help of the global RO and retrieve the message associated with his choice bit c . Intuitively, Property 2 now prevents the sender from learning the choice bit, while Property 1 ensures that the receiver learns at most one of the inputs.

While this simple protocol intuitively implements a stand alone oblivious transfer, it is hard to construct a simulator to prove it UC-secure in the global RO model. If programming the RO was allowed, the simulator could program the answer of the RO to a query s in such a way that it knows the secret keys corresponding to both pk_0 and pk_1 , allowing it to extract the messages from a corrupted sender. In the case of a corrupted receiver, the simulator could wait for the RO to be queried on one of the one-time pad seeds (extracting the choice bit), retrieve the message associated to that choice bit and program the answer of this RO query in such a way that the one-time pad encryption related to that seed decrypts to the message obtained from the OT functionality. However, the global RO model precludes us from using any of these techniques. Instead, we develop novel techniques for extracting both a corrupted receiver's choice bit and a corrupted sender's messages solely by observing global RO queries.

OT with Selective Failures: As a starting point, we design a protocol that UC-realizes a weaker version of the OT functionality, which captures a selective failure attack. This attack allows a malicious sender to try and “guess” the receiver's choice bit, only being caught if her guess is wrong. Allowing this selective failure makes it easier to implement mechanisms used by the simulator to extract the choice bit from a malicious receiver without the need to program the random oracle. Even though this protocol has a selective failure issue, it has been shown in [21] that it is sufficient to instantiate efficient OT extension protocols such as the one of [34]. Many applications require such a high number of oblivious transfers that it makes sense to use an actual OT protocol only to seed an OT extension, which can then be used for an unlimited number of OTs at very low cost. In order to simulate an execution with a corrupted receiver, we augment our simple protocol with a challenge-response mechanism inspired by [21] that forces the receiver to query the global RO in such a way that it reveals its choice bit to the simulator. In the real world protocol, the adversary can mount a selective failure attack where it can “guess” the receiver's choice bit, being caught if it guesses the wrong bit. However, a simulator who can observe the queries made to the global RO can easily determine the receiver's choice bit without resorting to a selective failure attack. This mechanism works by having the sender pick two random values $\mathbf{p}_0, \mathbf{p}_1$, compute a challenge $\text{ch} = \text{H}(\text{H}(\mathbf{p}_0)) + \text{H}(\text{H}(\mathbf{p}_1))$ where $\text{H}(\cdot)$ is the random oracle and send this challenge to the receiver along with encryptions of $\mathbf{p}_0, \mathbf{p}_1$. The receiver decrypts \mathbf{p}_c corresponding to its choice bit and answers with $\text{chr} = \text{H}(\text{H}(\mathbf{p}_c)) + c \cdot \text{ch}$, which will always be $\text{H}(\text{H}(\mathbf{p}_0))$ when ch is computed correctly. After receiving chr , the sender provides the receiver with $\text{H}(\mathbf{p}_0)$ and $\text{H}(\mathbf{p}_1)$, so that it can check that ch was correctly computed and

that $H(p_c)$ is consistent with the value it decrypted. However, a malicious sender can always guess the receiver's choice bit and compute ch in such a way that it will learn the actual choice bit but only be caught if it guesses wrong. Due to Properties 1 and 3, the simulator can be assured that the query p_c done by the receiver corresponds to its choice bit. The case of a corrupted sender is handled by a novel technique where the sender is forced to query the global RO in a way that reveals both of its messages to a simulator who can observe RO queries. The basic idea is to modify the challenge-response mechanism by having the sender query the global RO not only with the challenge seed p_i but also adding the public-key pk_i , and randomness r_i used to encrypt p_i to the query. Using Property 5, the receiver can complete the challenge-response mechanism since it can recover r_i used in the encryption of p_i . Using Property 3, the simulator is assured that a malicious sender could only have generated one such query for each pair of value p_i and randomness r_i . Hence, the simulator can check which pairs r_i, p_i in the list of queries to the global RO results in the ciphertexts sent by the sender when used as input to an encryption under pk_i . After extracting both p_0, p_1 , the simulator detects whether the adversary is trying to guess the choice bit (as well as the bit being guessed), which it forwards to the functionality. Later on, the sender uses the same p_i and corresponding randomness r_i to query a different instance of the global RO and obtain a one-time pad used to encrypt the actual messages it wants to transfer. Hence, the simulator can also use p_0, p_1 to extract both messages transferred by a malicious sender.

Eliminating Selective Failures: We are also interested in solving the problem of directly UC-realizing a standard OT functionality in the observable global random oracle model. In order to do so, we must eliminate the selective failure issue of our first protocol. We observe that we can do so by basically running two instances of our first protocol in parallel with the same public-keys pk_0 and pk_1 . Notice that these public-keys encode the choice bit, meaning that the same choice bit is used in both instances. The first instance will be used to extract the receiver's choice bit while ensuring a malicious sender cannot learn it through a selective failure attack. The other instance will be used to execute an oblivious transfer with the previously extracted choice bit and random messages, which can be later derandomized through standard techniques. We will run both protocol instances with a random choice bit, so that the receiver's actual choice bit does not leak in case the sender mounts a selective failure attack, which will be detected causing the execution to abort. In one of these instances, we will execute the challenge-response mechanism with the additional requirement that the sender must reveal both p_0, r_0 and p_1, r_1 , allowing the receiver to be sure no selective failure attack occurred. With this instance we are able to extract the receiver's random choice bit while ensuring that in the second instance the same bit will be used (because it is encoded in the keys pk_0 and pk_1 , also used in the second instance). In the second instance, we do not execute the challenge-response mechanism but use pk_0 and pk_1 to encrypt a second pair of seeds \hat{p}_0, \hat{p}_1 with randomness r'_0, r'_1 , which the sender queries to another instance of the global RO to obtain one-time pads for random messages being transferred. Due to Prop-

erty 3, the simulator can extract \hat{p}_0, \hat{p}_1 from the queries to the global RO and retrieve these random messages. At this point we have executed a random oblivious transfer, which is then derandomized to the receiver’s actual choice bit and the sender’s actual messages using standard information theoretical techniques.

2 Preliminaries

We denote by κ the security parameter. Let $y \stackrel{\$}{\leftarrow} F(x)$ denote running the randomized algorithm F with input x and random coins, and obtaining the output y . When the coins r are specified we use $y \leftarrow F(x; r)$. Similarly, $y \leftarrow F(x)$ is used for a deterministic algorithm. For a set \mathcal{X} , let $x \stackrel{\$}{\leftarrow} \mathcal{X}$ denote x chosen uniformly at random from \mathcal{X} ; and for a distribution \mathcal{Y} , let $y \stackrel{\$}{\leftarrow} \mathcal{Y}$ denote y sampled according to the distribution \mathcal{Y} . We will denote by $\text{negl}(\kappa)$ the set of negligible functions of κ . We abbreviate *probabilistic polynomial time* as PPT.

Encryption Schemes: The main building block used in our OT protocol is a public-key encryption scheme PKE. It has public-key \mathcal{PK} , secret-key \mathcal{SK} , message \mathcal{M} , randomness \mathcal{R} and ciphertext \mathcal{C} spaces that are functions of the security parameter κ , and consists of a PPT key generation algorithm KG, a PPT encryption algorithm Enc and a deterministic decryption algorithm Dec. For $(pk, sk) \stackrel{\$}{\leftarrow} \text{KG}(1^\kappa)$, any $m \in \mathcal{M}$, and $c \stackrel{\$}{\leftarrow} \text{Enc}(pk, m)$, it should hold that $\text{Dec}(sk, ct) = m$ with overwhelming probability over the used randomness.

We should emphasize that for some encryption schemes not all $\tilde{pk} \in \mathcal{PK}$ are “valid” in the sense of being a possible output of KG. The same holds for $\tilde{ct} \in \mathcal{C}$ in relation to Enc and all possible coins and messages. Our OT protocol uses as a building block a PKE that satisfies a variant of the OW-CPA security notion: informally, two random messages are encrypted under two different public-keys, one of which can be chosen by the adversary (but he does not have total control over both public-keys). His goal is then to recover both messages and this should be difficult. Formally, this property is captured by the following definition.

Property 1 (Double OW-CPA Security). Consider the public-key encryption scheme PKE and the security parameter κ . It is assumed that \mathcal{PK} forms a group with operation denoted by “ \star ”. For every PPT two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ running the following experiment:

$$\begin{aligned} q &\stackrel{\$}{\leftarrow} \mathcal{PK} \\ (pk_0, pk_1, st) &\stackrel{\$}{\leftarrow} \mathcal{A}_1(q) \text{ such that } pk_0, pk_1 \in \mathcal{PK} \text{ and } pk_0 \star pk_1 = q \\ m_i &\stackrel{\$}{\leftarrow} \mathcal{M} \text{ for } i = 0, 1 \\ ct_i &\stackrel{\$}{\leftarrow} \text{Enc}(pk_i, m_i) \text{ for } i = 0, 1 \\ (\tilde{m}_0, \tilde{m}_1) &\stackrel{\$}{\leftarrow} \mathcal{A}_2(ct_0, ct_1, st) \end{aligned}$$

it holds that

$$\Pr[(\tilde{m}_0, \tilde{m}_1) = (m_0, m_1)] \in \text{negl}(\kappa).$$

We also need a property about the indistinguishability of a public-key generated using KG and an element sampled uniformly at random from \mathcal{PK} .

Property 2 (Pseudorandomness of Public-Keys). Consider the public-key encryption scheme PKE and the security parameter κ . Let $(pk, sk) \xleftarrow{\$} \text{KG}(1^\kappa)$ and $pk' \xleftarrow{\$} \mathcal{PK}$. For every PPT distinguisher \mathcal{A} , it holds that

$$|\Pr[\mathcal{A}(pk) = 1] - \Pr[\mathcal{A}(pk') = 1]| \in \text{negl}(\kappa).$$

Moreover, we need the PKE scheme to be committing, meaning that an adversary can only generate two different pairs of randomness and plaintext message that result in the same ciphertexts when encrypted under a uniformly random public-key with negligible probability.

Property 3 (Committing Encryption). Consider the public-key encryption scheme PKE and the security parameter κ . For every PPT adversary \mathcal{A} , it holds that:

$$\Pr \left[\text{Enc}(pk, m_0; r_0) = \text{Enc}(pk, m_1; r_1) \left| \begin{array}{l} pk \xleftarrow{\$} \mathcal{PK}, \\ (r_0, r_1, m_0, m_1) \xleftarrow{\$} \mathcal{A}(pk), \\ r_0, r_1 \in \mathcal{R}, m_0, m_1 \in \mathcal{M}, \\ m_0 \neq m_1 \end{array} \right. \right] \in \text{negl}(\kappa)$$

Note that if Properties 2 and 3 hold for some PKE, then the modified version of Property 3 in which pk is chosen using KG also trivially holds. Moreover, we will need a variation of the committing property stating that even if an adversary is allowed to provide an arbitrary secret and public-key pair, it cannot both decrypt a ciphertext generated under that public-key *and* break the standard committing property. The rationale behind this property is that, for some committing encryption schemes, an adversary can generate an arbitrary public-key that breaks the standard committing property. However, in most cases, such a public-key will also cause plaintext information to be lost, making it impossible for the adversary to recover the original message from a ciphertext encrypted under this key with probability 1. This property is formalized in Property 4.

Property 4 (Committing Encryption with Arbitrary Keys). Consider the public-key encryption scheme PKE and the security parameter κ . For every PPT two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ running the following experiment:

$$\begin{aligned} (pk, st) &\xleftarrow{\$} \mathcal{A}_1(1^\kappa) \\ m &\xleftarrow{\$} \mathcal{M}, r \xleftarrow{\$} \mathcal{R} \\ ct &\leftarrow \text{Enc}(pk, m; r) \\ ((m', r'), (m_1, r_1), \dots, (m_{n-1}, r_{n-1})) &\xleftarrow{\$} \mathcal{A}_2(ct, st) \end{aligned}$$

it holds that

$$\Pr[m' = m \wedge r' = r \wedge (m_i, r_i) \neq (m, r) \wedge ct \leftarrow \text{Enc}(pk, m_i, r_i) \forall i = 1, \dots, n-1] \leq \frac{1}{n} + \text{negl}(\kappa).$$

We require PKE to have a *witness-recovering* decryption algorithm. Informally, this property means that the decryption algorithm also recovers the randomness used to generate the ciphertext it takes as input. Witness-recovering decryption is formally defined in Property 5.

Property 5 (Witness-Recovering Decryption). A public-key encryption scheme $\text{PKE} = (\text{KG}, \text{Enc}, \text{Dec})$ has a witness-recovering decryption algorithm Dec if it takes as input the secret-key $\text{sk} \in \mathcal{SK}$ and a ciphertext $\text{ct} \in \mathcal{C}$ and outputs either a pair (m, r) for $\text{m} \in \mathcal{M}$ and $\text{r} \in \mathcal{R}$ or an error symbol \perp . For any $(\text{pk}, \text{sk}) \xleftarrow{\$} \text{KG}(1^\kappa)$, any $\text{m} \in \mathcal{M}$, any $\text{r} \xleftarrow{\$} \mathcal{R}$ and $c \leftarrow \text{Enc}(\text{pk}, \text{m}; \text{r})$, it should hold that $\text{Dec}(\text{sk}, \text{ct}) = (\text{m}, \text{r})$ with overwhelming probability over the randomness used by the algorithms.

In the full version [17], we prove that Properties 1, 2, 3 and 4 hold for the ElGamal cryptosystems based on the CDH assumption, yielding an efficient instantiation of our generic protocol. Even though the ElGamal cryptosystem does not have a straightforward witness-recovery decryption algorithm, we show how any OW-CPA secure public-key encryption scheme used on random messages can be augmented with such a decryption algorithm to achieve Property 5. This can be done through the encrypt-with-hash paradigm, where the randomness used for encryption is obtained by hashing the message being encrypted, which can be proven secure in the non-programmable random oracle model.

Functionality \mathcal{F}_{gRO}

\mathcal{F}_{gRO} is parameterized by a range \mathcal{D} and a list of ideal functionalities $\overline{\mathcal{F}}$.

- Upon receiving a query x from some party $P = (\text{pid}, \text{sid})$ or from the adversary \mathcal{S} do:
 - If there is a pair (x, v) for some $v \in \mathcal{D}$ in the (initially empty) list \mathcal{Q} of past queries, return v to P . Else, sample $v \xleftarrow{\$} \mathcal{D}$ and store the pair (x, v) in \mathcal{Q} . Return v to P .
 - Parse x as (s, x') . If $\text{sid} \neq s$, then add (s, x', v) to the (initially empty) list of illegitimate queries for SID s , denoted by $\mathcal{Q}_{|s}$.
- Upon receiving a request from an instance of an ideal functionality in the list $\overline{\mathcal{F}}$, with SID s , return to this instance the list $\mathcal{Q}_{|s}$ of illegitimate queries for SID s .

Fig. 1. Functionality \mathcal{F}_{gRO} .

Universal Composability in the Global Random Oracle Model: We analyze our protocol in the UC model with global random oracles as presented in [10]. We refer interested readers to the original work for more details on the UC framework [7]. In the UC model with global random oracles, the parties are assumed to have access to a global random oracle functionality \mathcal{F}_{gRO} (see Fig. 1 for details) and interfaces that leak the list of illegitimate queries $\mathcal{Q}_{|s}$ to the adversary. Differently from the basic UC model, the global random oracle model allows all parties (including the environment) to access a single instance of \mathcal{F}_{gRO} . The \mathcal{F}_{gRO} functionality functions as a regular random oracle but is augmented

with a mechanism for leaking queries performed by parties that are not part of a given execution. In the UC model parties are identified by a unique pair of program id (PID) and session id (SID). Queries that are not prepended with the same SID as the one identifying the party $P = (\text{pid}, \text{sid})$ making the query are added to a list of illegitimate queries that can be requested by instances of functionalities whose session id match the one in the query. This mechanism allows the simulator to learn queries made by the environment or adversary but keeps the queries made by honest parties secret (as honest parties will follow the protocol and prepend their queries with the correct SID). Moreover, the functionalities in the global random oracle model take into consideration the existence of this list of illegitimate queries, requesting it from \mathcal{F}_{gRO} and handing it to the adversary, if requested by the adversary. Our construction will actually use three instances of \mathcal{F}_{gRO} : $\mathcal{F}_{\text{gRO}1}$ with range \mathcal{PK} , $\mathcal{F}_{\text{gRO}2}$ with range $\{0, 1\}^\lambda$ and $\mathcal{F}_{\text{gRO}3}$ with range $\{0, 1\}^\kappa$.

We consider a static malicious adversary. I.e., it can deviate from the prescribed protocol in an arbitrary way, but has to corrupt the parties before the execution starts.

Functionality $\mathcal{F}_{\text{SFOT}}^\lambda$.

$\mathcal{F}_{\text{SFOT}}^\lambda$ is parameterized by the length of the messages $\lambda \in \mathbb{N}$, which is publicly known. $\mathcal{F}_{\text{SFOT}}^\lambda$ interacts with a sender Alice and a receiver Bob, proceeding as follows:

- Upon receiving a message (choose, sid, c) from Bob, where $c \in \{0, 1\}$, record (sid, choice, c), send (chosen, sid) to Alice, and ignore future messages (choose, sid, \cdot) with the same sid.
- Upon receiving a message (guess, sid, \hat{c}) from Alice, where $\hat{c} \in \{0, 1, \perp, \text{force}\}$, if a tuple (sid, choice, c) is recorded, then record (sid, guess, \hat{c}), ignore future messages (guess, sid, \cdot) with the same sid and do the following:
 1. If $\hat{c} = \perp$, send (no – cheat, sid) to Bob.
 2. If $\hat{c} = c$, send (cheat – undetected, sid) to Alice and (no – cheat, sid) to Bob.
 3. If $\hat{c} \neq c$ or $\hat{c} = \text{force}$, send (cheat – detected, sid, c) to both Alice and Bob.
- Upon receiving a message (send, sid, $\mathbf{x}_0, \mathbf{x}_1$) from Alice, where each $\mathbf{x}_i \in \{0, 1\}^\lambda$, if there are tuples (sid, choice, c) and (sid, guess, \hat{c}) recorded such that $\hat{c} = \perp$ or $\hat{c} = c$, then send (output, sid, \mathbf{x}_c) to Bob and ignore further messages from Alice with the same sid.
- When asked by \mathcal{S} , obtain from \mathcal{F}_{gRO} the list $\mathcal{Q}_{|\text{sid}}$ of illegitimate queries for SID sid and send it to \mathcal{S} .

Fig. 2. Functionality $\mathcal{F}_{\text{SFOT}}^\lambda$ in the global random oracle model.

Oblivious Transfer: The functionality $\mathcal{F}_{\text{OT}}^{\lambda, \ell}$ that provides ℓ instances of the 1-out-of-2 string (of length λ) oblivious transfer in the \mathcal{F}_{gRO} -hybrid model is

presented in the full version [17]. This work focus on obtaining a weaker form of oblivious transfer that allows selective failure attacks, aiming for the same type of weaker OT as in Doerner et al. [21]. The ideal functionality $\mathcal{F}_{\text{SFOT}}^\lambda$ for 1-out-of-2 string oblivious transfer with selective failure in the \mathcal{F}_{gRO} -hybrid model is described in Fig. 2. Essentially, the sender is given the option of trying to guess the choice bit of the receiver. If she makes a wrong guess, the cheating is detected and the execution aborts. If she makes a right guess, she learns the choice bit and nothing is detected by the receiver. As proved by Doerner et al. in the full version of their work [20], $\mathcal{F}_{\text{SFOT}}^\lambda$ can be used as the base OTs in the OT extension protocol of Keller et al. [34] to UC-realize $\mathcal{F}_{\text{OT}}^{\lambda,\ell}$.

Lemma 1. *The OT extension protocol of Keller et al. [34] UC-realizes $\mathcal{F}_{\text{OT}}^{\lambda,\ell}$ in the $\mathcal{F}_{\text{SFOT}}^\lambda, \mathcal{F}_{\text{gRO}}$ -hybrid model.*

Proof. This follows directly from Lemma D.3 of [20], which proves that the first part of the OT extension protocol UC-realizes the correlated OT with errors functionality $\mathcal{F}_{\text{COTe}}$ in the $\mathcal{F}_{\text{SFOT}}^\lambda, \mathcal{F}_{\text{gRO}}$ -hybrid model, and the reduction from $\mathcal{F}_{\text{OT}}^{\lambda,\ell}$ to $\mathcal{F}_{\text{COTe}}$ using the remaining steps of the OT extension protocol [34].

3 The Generic Protocol

Our protocol uses as a building block a public-key encryption scheme that satisfies Properties 1, 2, 3, 4 and 5 (defined in Sect. 2). The basic high-level idea is that Bob picks two public-keys pk_0, pk_1 such that he only knows the secret-key corresponding to pk_c (where c is his choice bit) and hands them to Alice. She then uses the two public-keys to transmit two messages in an encrypted way, so that Bob can only recover the message for which he knows the secret-key sk_c .

A crucial point in such schemes is making sure that Bob is only able to decrypt one of the messages. In order to enforce this property, our protocol relies on Property 1 and uses the random oracle to force the element q to be chosen uniformly at random from \mathcal{PK} . After generating the pair of public and secret-key $(\text{pk}_c, \text{sk}_c)$, Bob samples a seed s , queries the random oracle $\mathcal{F}_{\text{gRO1}}$ with s to obtain q , and computes pk_{1-c} such that $\text{pk}_0 \star \text{pk}_1 = q$. Bob then hands the public-key pk_0 and the seed s to Alice, enabling her to also compute pk_1 . Since the public-keys are indistinguishable according to Property 2, Alice learns nothing about Bob's choice bit. Next, Alice picks two uniformly random strings p_0, p_1 , queries them to the random oracle $\mathcal{F}_{\text{gRO2}}$ obtaining $\tilde{\text{p}}_0, \tilde{\text{p}}_1$ as response, and then she computes one-time pad encryptions of her messages m_0, m_1 as $\tilde{\text{m}}_0 = \text{m}_0 \oplus \tilde{\text{p}}_0$ and $\tilde{\text{m}}_1 = \text{m}_1 \oplus \tilde{\text{p}}_1$. Alice also computes $\text{ct}_0 \leftarrow \text{Enc}(\text{pk}_0, \text{p}_0; r_0)$, $\text{ct}_1 \leftarrow \text{Enc}(\text{pk}_1, \text{p}_1; r_1)$ and sends $(\tilde{\text{m}}_0, \tilde{\text{m}}_1, \text{ct}_0, \text{ct}_1)$ to Bob. Bob can use sk_c to decrypt ct_c obtaining p_c . He then queries p_c to the random oracle $\mathcal{F}_{\text{gRO2}}$ obtaining $\tilde{\text{p}}_c$ as response, and retrieves $\text{m}_c = \tilde{\text{m}}_c \oplus \tilde{\text{p}}_c$. Due to Property 1, Bob will not be able to recover p_{1-c} in order to query it to the random oracle and to decrypt $\tilde{\text{m}}_{1-c}$. Therefore, the security for Alice is also guaranteed.

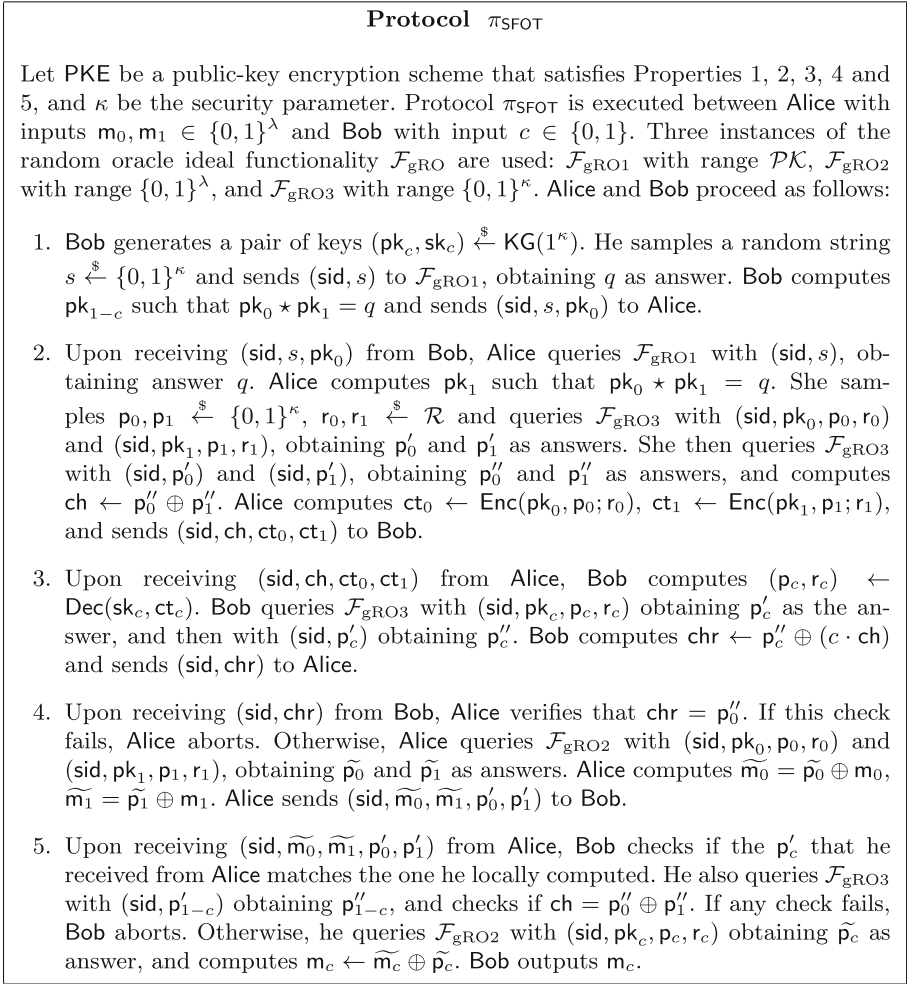


Fig. 3. Protocol π_{SFOT}

Even though this simple protocol seemingly performs an oblivious transfer, it poses significant challenges for a proof in the Global Random Oracle model of Canetti et al. [10], where the simulator cannot program the answers to random oracle queries. In the case of a malicious sender, the simulator would need to generate a seed s and public key pk_0 such that it knows both secret keys associated to the resulting public keys pk_0 and pk_1 , which it needs to know in order to extract the messages m_0 and m_1 . However, while this is easy if the simulator could program an arbitrary random oracle answer given the seed s , it cannot be done in this model. In the case of a malicious receiver, Property 2 ensures that the simulator cannot learn any information about the choice bit c before the adversary queries the random oracle on p_c , which only happens *after* the

simulator has sent its last message. The simulator could possibly program the random oracle answer given p_c so that the result is m_c (received from the OT functionality), but this is not possible in this setting. In order to circumvent these challenges, we augment the simple protocol described before with mechanisms that allow the simulator to extract the choice bit c and messages m_0 and m_1 without resorting to programming the random oracle.

In order to obtain security against a malicious receiver, we use a challenge-response mechanism that follows the approach of Doerner et al. [21]. Basically, before carrying out the actual transfer, Alice queries (pk_0, p_0, r_0) and (pk_1, p_1, r_1) to the random oracle \mathcal{F}_{gRO3} (note that this oracle is different from \mathcal{F}_{gRO2}) obtaining p'_0, p'_1 , and then queries p'_0, p'_1 to the random oracle \mathcal{F}_{gRO3} obtaining p''_0, p''_1 . Alice fixes the challenge as $ch \leftarrow p''_0 \oplus p''_1$ and sends ch to Bob. Bob queries \mathcal{F}_{gRO3} with (pk_c, p_c, r_c) , which is possible because PKE has witness-recovering decryption according to Property 5, obtaining p'_c and then with p'_c obtaining p''_c . Bob returns $p'_c \oplus (c \cdot ch)$ to Alice, who checks if the returned value is equal to p''_0 . Alice then sends p'_0, p'_1 to Bob, who checks if these values are compatible with the values he previously computed and ch . After receiving a valid response from Bob, Alice proceeds with the transfer. A crucial aspect of this mechanism is that in order to obtain p''_c , Bob is forced to first issue a query associated to its choice bit c to the random oracle, allowing for extraction. In the proof, the simulator can extract c solely by observing the adversary's queries after it receives the challenge, allowing it to obtain m_c from the OT functionality and prepare the last message to the adversary accordingly. This mechanism allows selective failure attacks, but the resulting scheme fulfills the requirements to be used as base OTs in the OT extension scheme of Keller et al. [34] (see Sect. 2).

Instead of querying \mathcal{F}_{gRO2} with p_0, p_1 , we query it with (pk_0, p_0, r_0) and (pk_1, p_1, r_1) to obtain \tilde{p}_0, \tilde{p}_1 . These queries of the form (pk_i, p_i, r_i) to \mathcal{F}_{gRO2} and \mathcal{F}_{gRO3} allow the simulator to extract both of the corrupt sender's messages solely by observing the queries to the random oracle. In the simulation, the simulator reconstructs ciphertexts $\hat{ct}_j = \text{Enc}(pk_i, \hat{p}_j, \hat{r}_j)$ from all random oracle queries of the form $(pk_i, \hat{p}_j, \hat{r}_j)$, looking for a ciphertext \hat{ct}_j that matches ciphertext ct_i (for $i \in \{0, 1\}$) in the adversary's message. Having found these ciphertexts the simulator can proceed to recover each message m_i . An adversary could try to confuse the simulator by making two different queries to the random oracle that pass the tests above. However, this is not possible due to Properties 3 and 4.

Protocol π_{SFOT} is described in Fig. 3 and its security is formally stated in Theorem 1, which we prove in the full version [17]. A CDH based instantiation is described in the full version [17].

Theorem 1. *Let PKE be a public-key encryption scheme that satisfies Properties 1, 2, 3, 4 and 5. When instantiated with PKE, Protocol π_{SFOT} UC-realizes functionality $\mathcal{F}_{\text{SFOT}}^\lambda$ with security against static malicious adversaries in the global random oracle model.*

4 Realizing $\mathcal{F}_{\text{OT}}^{\lambda,1}$ Directly

Our previous generic protocol can be modified to directly realize the standard 1-out-of-2 OT functionality $\mathcal{F}_{\text{OT}}^{\lambda,1}$ without any selective failure issues, instead of first realizing $\mathcal{F}_{\text{SFOT}}^{\lambda}$ and then employing the OT extension of Keller *et al.* to realize $\mathcal{F}_{\text{OT}}^{\lambda,\ell}$. However, we will rely directly on the specific CDH based PKE constructed in the full version [17] instead of a generic PKE with Properties 1, 2, 3, 4 and 5. This is necessary since the simulator will now need to extract messages encrypted under this PKE that it cannot extract by simply observing queries to the random oracle instances used in the protocol but that can be extracted by observing queries to the random oracle instance used by this specific PKE construction.

In order to eliminate the potential selective failure from our first protocol, we need to provide **Bob** with a proof that **Alice** has used exactly the values $\mathbf{p}_0, \mathbf{p}_1$ contained in ciphertexts ct_0, ct_1 to generate challenge ch . The main idea is to use two instances of our original protocol that are run using the same public keys $\mathbf{pk}_0, \mathbf{pk}_1$ (encoding the same choice bit). One of them is used to execute the challenge-response mechanism and the other is used to execute a random OT, which can be later derandomized. In our previous protocol, **Alice** only reveals the outputs of $\mathcal{F}_{\text{gRO3}}$ upon being queried with $(\text{sid}, \mathbf{pk}_i, \mathbf{p}_i, r_i)$, which only allows **Bob** to check that these were the values used in the challenge with probability $\frac{1}{2}$. In order to prove that those values were indeed used, we will leverage the committing property (Property 3) of the underlying cryptosystem and have **Alice** reveal $\mathbf{p}_0, \mathbf{p}_1, r_0, r_1$ to **Bob** upon getting a valid response to the challenge. Using these values, **Bob** can recompute the challenge (checking that it matches ch received from **Alice**) and check that $\text{ct}_i \stackrel{\$}{\leftarrow} \text{Enc}(\mathbf{pk}_i, \mathbf{p}_i; r_i)$, for $i = \{0, 1\}$. If those checks fail, the receiver aborts but, if they succeed, it is assured by the committing property that those values were used in computing ct_0, ct_1 and ch (meaning the choice bit was not leaked). Having both $\mathbf{p}_0, \mathbf{p}_1$ revealed to **Bob**, we will need to have **Alice** generate new $\hat{\mathbf{p}}_0, \hat{\mathbf{p}}_1$ and corresponding $\hat{\text{ct}}_0, \hat{\text{ct}}_1$ to complete the OT as in our first protocol. However, notice that this protocol still leaks **Bob**'s choice bit to an adversary who mounts a successful selective failure attack, even though the attack is detected and the protocol is aborted. In order to deal with this, **Bob** uses a random choice bit to execute a random OT that is derandomized after **Bob** is certain no selective failure attack occurred.

The simulator for a corrupt **Alice** does not have to extract the “guess” bit of the adversary, just acting as an honest **Bob** and extracting the messages $\mathbf{m}_0, \mathbf{m}_1$ using the same techniques as the simulator in π_{SFOT} . However, it will need to extract messages $\hat{\mathbf{p}}_0, \hat{\mathbf{p}}_1$ from the ciphertexts $\hat{\text{ct}}_0, \hat{\text{ct}}_1$ by observing queries to the random oracle used in the CDH based PKE described in the full version [17]. The simulator for a corrupt **Bob** uses the same techniques as the simulator in π_{SFOT} to extract the choice bit. The difference is that the ciphertexts $\text{ct}'_0, \text{ct}'_1$ obtained from the challenger in the game of Property 1 are given to the adversary as $\text{ct}_c, \hat{\text{ct}}_{1-c}$ in the reduction showing that an adversary that obtains \mathbf{m}_{1-c} when interacting with this simulator breaks Property 1.

Protocol π_{OT}

Let PKE be the CDH based public-key encryption scheme described in the full version [17] that satisfies Properties 1, 2, 3, 4 and 5, and κ be the security parameter. Protocol π_{OT} is executed between Alice with inputs $m_0, m_1 \in \{0, 1\}^\lambda$ and Bob with input $c \in \{0, 1\}$. Bob and Alice interact with each other and with four instances of the random oracle ideal functionality: $\mathcal{F}_{\text{gRO1}}$ with range \mathcal{PK} , $\mathcal{F}_{\text{gRO2}}$ with range $\{0, 1\}^\lambda$, $\mathcal{F}_{\text{gRO3}}$ with range $\{0, 1\}^\kappa$, and $\mathcal{F}_{\text{gRO4}}$ with range \mathcal{R} (used by PKE). Protocol π_{OT} proceeds as follows:

1. Bob samples $c' \xleftarrow{\$} \{0, 1\}$ and generates a pair of keys $(\text{pk}_{c'}, \text{sk}_{c'}) \xleftarrow{\$} \text{KG}(1^\kappa)$. He samples a random string $s \xleftarrow{\$} \{0, 1\}^\kappa$ and sends (sid, s) to $\mathcal{F}_{\text{gRO1}}$, obtaining q as answer. Bob computes $\text{pk}_{1-c'}$ such that $\text{pk}_0 \star \text{pk}_1 = q$ and sends $(\text{sid}, s, \text{pk}_0)$ to Alice.
2. Upon receiving $(\text{sid}, s, \text{pk}_0)$ from Bob, Alice queries $\mathcal{F}_{\text{gRO1}}$ with (sid, s) , obtaining answer q and computing pk_1 such that $\text{pk}_0 \star \text{pk}_1 = q$. For $i \in \{0, 1\}$, Alice samples $\text{p}_i, \hat{\text{p}}_i \xleftarrow{\$} \{0, 1\}^\kappa$ and $r_i, \hat{r}_i \xleftarrow{\$} \mathcal{R}$, queries $\mathcal{F}_{\text{gRO3}}$ with $(\text{sid}, \text{pk}_i, \text{p}_i, r_i)$, obtaining p'_i as answer, and then queries $\mathcal{F}_{\text{gRO3}}$ with $(\text{sid}, \text{p}'_i)$, obtaining p''_i as answer. Alice computes $\text{ch} \leftarrow \text{p}''_0 \oplus \text{p}''_1$. For $i \in \{0, 1\}$, Alice computes $\text{ct}_i \leftarrow \text{Enc}(\text{pk}_i, \text{p}_i; r_i)$, $\hat{\text{ct}}_i \leftarrow \text{Enc}(\text{pk}_i, \hat{\text{p}}_i; \hat{r}_i)$. Alice sends $(\text{sid}, \text{ch}, \text{ct}_0, \text{ct}_1, \hat{\text{ct}}_0, \hat{\text{ct}}_1)$ to Bob.
3. Upon receiving $(\text{sid}, \text{ch}, \text{ct}_0, \text{ct}_1, \hat{\text{ct}}_0, \hat{\text{ct}}_1)$ from Alice, Bob computes $(\text{p}_{c'}, r_{c'}) \leftarrow \text{Dec}(\text{sk}_{c'}, \text{ct}_{c'})$, queries $\mathcal{F}_{\text{gRO3}}$ with $(\text{sid}, \text{pk}_{c'}, \text{p}_{c'}, r_{c'})$, obtaining $\text{p}'_{c'}$ as the answer, and then queries $\mathcal{F}_{\text{gRO3}}$ with $(\text{sid}, \text{p}'_{c'})$, obtaining $\text{p}''_{c'}$ as the answer. Bob computes $\text{chr} \leftarrow \text{p}''_{c'} \oplus (c' \cdot \text{ch})$ and sends (sid, chr) to Alice.
4. Upon receiving (sid, chr) from Bob, Alice verifies that $\text{chr} = \text{p}''_0$. If this check fails, Alice aborts. Otherwise, for $i \in \{0, 1\}$, Alice queries $\mathcal{F}_{\text{gRO2}}$ with $(\text{sid}, \text{pk}_i, \hat{\text{p}}_i, \hat{r}_i)$ (obtaining $\tilde{\text{p}}_i$ as answer), samples $\hat{m}_i \xleftarrow{\$} \{0, 1\}^\lambda$ and computes $\tilde{m}_i = \tilde{\text{p}}_i \oplus \hat{m}_i$. Alice sends $(\text{sid}, \tilde{m}_0, \tilde{m}_1, \text{p}_0, \text{p}_1, r_0, r_1)$ to Bob.
5. Upon receiving $(\text{sid}, \tilde{m}_0, \tilde{m}_1, \text{p}_0, \text{p}_1, r_0, r_1)$ from Alice, for $i \in \{0, 1\}$, Bob checks that $\text{ct}_i = \text{Enc}(\text{pk}_i, \text{p}_i, r_i)$ and queries $\mathcal{F}_{\text{gRO3}}$ with $(\text{sid}, \text{pk}_i, \text{p}_i, r_i)$ (obtaining p'_i as answer) and queries $\mathcal{F}_{\text{gRO3}}$ with $(\text{sid}, \text{p}'_i)$ (obtaining p''_i as answers). Next Bob checks that $\text{ch} = \text{p}''_0 \oplus \text{p}''_1$. If any checks fail, Bob aborts. Otherwise, Bob computes $(\hat{\text{p}}_{c'}, \hat{r}_{c'}) \leftarrow \text{Dec}(\text{sk}_{c'}, \hat{\text{ct}}_{c'})$. If this decryption fails with $\perp \leftarrow \text{Dec}(\text{sk}_{c'}, \hat{\text{ct}}_{c'})$, Bob samples a random $(\hat{\text{p}}_{c'}, \hat{r}_{c'}) \xleftarrow{\$} \{0, 1\}^\kappa \times \mathcal{R}$ in order to avoid a selective failure. Bob queries $\mathcal{F}_{\text{gRO2}}$ with $(\text{sid}, \text{pk}_{c'}, \hat{\text{p}}_{c'}, \hat{r}_{c'})$, obtaining $\tilde{\text{p}}_{c'}$ as answer, computes $\hat{m}_{c'} \leftarrow \tilde{m}_{c'} \oplus \tilde{\text{p}}_{c'}$, sets $d \leftarrow c \oplus c'$ and sends (sid, d) to Alice.
6. Upon receiving (sid, d) from Bob, Alice sets $m'_0 \leftarrow \hat{m}_d \oplus m_0, m'_1 \leftarrow \hat{m}_{1-d} \oplus m_1$. Alice sends (sid, m'_0, m'_1) to Bob.
7. Upon receiving (sid, m'_0, m'_1) from Alice, Bob computes $m_c = m'_c \oplus \hat{m}_{c'}$, outputs m_c and halts.

Fig. 4. Protocol π_{OT}

Protocol π_{OT} is described in Fig. 4 and its security is formally stated in Theorem 2, which we prove in the full version [17]. The CDH based PKE instantiation is described in the full version [17].

Theorem 2. *Under the CDH assumption, Protocol π_{OT} UC-realizes functionality $\mathcal{F}_{\text{OT}}^{\lambda,1}$ with security against static malicious adversaries in the global random oracle model.*

References

1. Barreto, P.S.L.M., David, B., Dowsley, R., Morozov, K., Nascimento, A.C.A.: A framework for efficient adaptively secure composable oblivious transfer in the rom. Cryptology ePrint Archive, Report 2017/993 (2017). <https://eprint.iacr.org/2017/993>
2. Bellare, M., Micali, S.: Non-interactive oblivious transfer and applications. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 547–557. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_48
3. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Ashby, V. (ed.) ACM CCS 1993, pp. 62–73. ACM Press, November 1993
4. Brzuska, C., Fischlin, M., Schröder, H., Katzenbeisser, S.: Physically uncloneable functions in the universal composition framework. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 51–70. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_4
5. Byali, M., Patra, A., Ravi, D., Sarkar, P.: Efficient, round-optimal, universally-composable oblivious transfer and commitment scheme with adaptive security. Cryptology ePrint Archive, Report 2017/1165 (2017). <https://eprint.iacr.org/2017/1165>
6. Camenisch, J., Drijvers, M., Gagliardini, T., Lehmann, A., Neven, G.: The wonderful world of global random oracles. In: Nielsen, J.B., Rijmen, V. (eds.) EURO-CRYPT 2018. LNCS, vol. 10820, pp. 280–312. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_11
7. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001
8. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 61–85. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_4
9. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_2
10. Canetti, R., Jain, A., Scafuro, A.: Practical UC security with a global random oracle. In: Ahn, G.-J., Yung, M., Li, N. (eds.), ACM CCS 2014, pp. 597–608. ACM Press, November 2014
11. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th ACM STOC, pp. 494–503. ACM Press, May 2002
12. Canetti, R., Sarkar, P., Wang, X.: Blazing fast OT for three-round UC OT extension. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12111, pp. 299–327. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45388-6_11

13. Choi, S.G., Katz, J., Wee, H., Zhou, H.-S.: Efficient, adaptively secure, and composable oblivious transfer with a single, global CRS. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 73–88. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_6
14. Chou, T., Orlandi, C.: The simplest protocol for oblivious transfer. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 40–58. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22174-8_3
15. Damgård, I., Nielsen, J.B.: Universally composable efficient multiparty computation from threshold homomorphic encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 247–264. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_15
16. Damgård, I., Nielsen, J.B., Orlandi, C.: Essentially optimal universally composable oblivious transfer. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 318–335. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00730-9_20
17. David, B., Dowsley, R.: Efficient composable oblivious transfer from CDH in the global random oracle model. Cryptology ePrint Archive, Report 2020/1291 (2020). <https://eprint.iacr.org/2020/1291>
18. David, B., Dowsley, R., Nascimento, A.C.A.: Universally composable oblivious transfer based on a variant of LPN. In: Gritzalis, D., Kiayias, A., Askoxylakis, I. (eds.) CANS 2014. LNCS, vol. 8813, pp. 143–158. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12280-9_10
19. David, B.M., Nascimento, A.C.A., Müller-Quade, J.: Universally composable oblivious transfer from lossy encryption and the McEliece assumptions. In: Smith, A. (ed.) ICITS 2012. LNCS, vol. 7412, pp. 80–99. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32284-6_5
20. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ECDSA from ECDSA assumptions. Cryptology ePrint Archive, Report 2018/499 (2018). <https://eprint.iacr.org/2018/499>
21. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ECDSA from ECDSA assumptions. In: 2018 IEEE Symposium on Security and Privacy, pp. 980–997. IEEE Computer Society Press, May 2018
22. Döttling, N., Garg, S., Hajiabadi, M., Masny, D., Wichs, D.: Two-round oblivious transfer from CDH or LPN. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 768–797. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_26
23. Döttling, N., Kraschewski, D., Müller-Quade, J.: Unconditional and composable security using a single stateful tamper-proof hardware token. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 164–181. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_11
24. Dowsley, R., Müller-Quade, J., Nascimento, A.C.A.: On the composability of statistically secure random oblivious transfer. Entropy **22**(1), 107 (2020)
25. Dowsley, R., Müller-Quade, J., Nilges, T.: Weakening the isolation assumption of tamper-proof hardware tokens. In: Lehmann, A., Wolf, S. (eds.) ICITS 2015. LNCS, vol. 9063, pp. 197–213. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17470-9_12
26. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM **28**(6), 637–647 (1985)

27. Friolo, D., Masny, D., Venturi, D.: A black-box construction of fully-simulatable, round-optimal oblivious transfer from strongly uniform key agreement. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 111–130. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_5
28. Garay, J.A., MacKenzie, P., Yang, K.: Efficient and universally composable committed oblivious transfer and applications. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 297–316. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_17
29. Genç, Z.A., Iovino, V., Rial, A.: The simplest protocol for oblivious transfer” revisited. Cryptology ePrint Archive, Report 2017/370 (2017). <https://eprint.iacr.org/2017/370>
30. Hauck, E., Loss, J.: Efficient and universally composable protocols for oblivious transfer from the CDH assumption. Cryptology ePrint Archive, Report 2017/1011 (2017). <http://eprint.iacr.org/2017/1011>
31. Hazay, C., Venkitasubramaniam, M.: On black-box complexity of universally composable security in the CRS model. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 183–209. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_8
32. Jarecki, S., Shmatikov, V.: Efficient two-party secure computation on committed inputs. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 97–114. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_6
33. Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_7
34. Keller, M., Orsini, E., Scholl, P.: Actively secure OT extension with optimal overhead. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 724–741. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_35
35. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Kosaraju, S.R. (eds.), 12th SODA, pp. 448–457. ACM-SIAM, January 2001
36. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_31
37. Rabin, M.O.: How to exchange secrets by oblivious transfer. Technical Report Technical Memo TR-81, Aiken Computation Laboratory, Harvard University (1981)

Lightweight Cryptography



Integral Cryptanalysis of Reduced-Round Tweakable TWINE

Muhammad ElSheikh and Amr M. Youssef^(✉)

Concordia Institute for Information Systems Engineering, Concordia University,
Montréal, QC, Canada
{m_elshei,youssef}@ciise.concordia.ca

Abstract. Tweakable TWINE (T-TWINE) is the first lightweight dedicated tweakable block cipher family built on Generalized Feistel Structure (GFS). T-TWINE family is an extension of the conventional block cipher TWINE with minimal modification by adding a simple tweak based on the SKINNY's tweak schedule. Similar to TWINE, T-TWINE has two variants, namely T-TWINE-80 and T-TWINE-128. The two variants have the same block size of 64 bits and a variable key length of 80 and 128 bits. In this paper, we study the implications for adding the tweak on the security of T-TWINE against the integral cryptanalysis. In particular, we first utilize the bit-based division property to search for the longest integral distinguisher. As a result, we are able to perform a distinguishing attack against 19 rounds using $2^6 \times 2^{63} = 2^{69}$ chosen tweak-plaintext combinations. We then convert this attack to key recovery attacks against 26 and 27 rounds (out of 36) of T-TWINE-80 and T-TWINE-128, respectively. By prepending one round before the distinguisher and using dynamically chosen plaintexts, we manage to extend the attack one more round without using the full codebook of the plaintext. Therefore, we are able to attack 27 and 28 rounds of T-TWINE-80 and T-TWINE-128, respectively.

1 Introduction

A Tweakable block cipher (TBC) is a symmetric-key cryptographic primitive that takes an auxiliary input called *tweak* in addition to the inputs of traditional block ciphers, plaintext message and cryptographic key [12]. Ideally, a different tweak value gives randomly chosen and different instant of the permutation over the message space without needing to change the key which may be costly in traditional block ciphers. A Tweakable block cipher is a powerful primitive that can be used in several applications such as disk encryption in which the repeated same plaintext should be encrypted to different ciphertexts under the same key. The concept of tweakable block ciphers also allows interesting modes for authenticated encryption such as OCB3 [11] and Counter-in-Tweak [13].

There are two general approaches to build TBCs: (i) using ordinary block ciphers through modes of operation, and (ii) dedicated constructions. Both the LRW and XEX modes of operations [14] are examples of the first approach. For

a block cipher with n -bit block, the security of these modes is guaranteed up to around $2^{n/2}$ queries. For a higher level of security, we can use a dedicated TBC that is built with the tweak concept from the beginning such as Deoxys-BC [9], SKINNY [1], and CRAFT [2].

Tweakable TWINE (T-TWINE) [15] is the first lightweight dedicated TBC that is built on Generalized Feistel Structure (GFS). It was built with the goal of reducing the cost of design, security evaluation, and implementation. Therefore, the designers decided to reuse a well-designed GFS block cipher, TWINE [20], and attached an extremely simple tweak scheduling to it. Similar to TWINE, T-TWINE has two variants namely, T-TWINE-80 and T-TWINE-128. These variants have the same block size of 64 bits, a tweak of 64 bits, and a variable key length of 80 and 128 bits.

The security of T-TWINE is evaluated by its designers against distinguishing attacks including differential, linear, impossible differential, and integral cryptanalysis. Regarding the integral cryptanalysis, they only reported an 11-round integral distinguisher. Key recovery attacks based on impossible differential against reduced-round of T-TWINE are presented in [22].

Our Contributions. In this work, we study the security of T-TWINE against the integral attack. More precisely,

1. We utilize a Mixed-Integer-Linear Programming (MILP) model of the bit-based division property to search for the longest integral distinguisher in the chosen tweak, chosen tweak-plaintext, and chosen tweak-ciphertext attack settings. As a result, we found two 11-round integral distinguishers using a tweak with only one active nibble in the chosen tweak setting. We also checked the 11-round distinguisher reported in the design paper and we show that it is not correct. All the found 11-round distinguishers are verified experimentally. Furthermore, we found several 19-round integral distinguishers in both chosen tweak-plaintext and chosen tweak-ciphertext settings. This allows us to attack an extra three rounds more than TWINE which has 16-round integral distinguisher [24]. The best distinguishing attack can be performed using $2^6 \times 2^{63} = 2^{69}$ chosen tweak-plaintext combinations.
2. We employ meet-in-the-middle [16] and partial-sum [7] techniques to convert the best distinguishing attack to key recovery attacks against 26 (27) out of 36 rounds of T-TWINE-80 (T-TWINE-128) by appending 7 (8) rounds after the distinguisher.
3. By prepending one round before the distinguisher and using dynamically chosen plaintexts [3], we managed to extend the attack one more round without using the full codebook of the plaintext. Therefore, we are able to attack 27 and 28 rounds of T-TWINE-80 and T-TWINE-128, respectively.

Table 1 summarizes the complexities of our attacks and contrast them with the complexities of the impossible differential attacks presented in [22].

Table 1. Attack results on T-TWINE where CTP denotes chosen tweak-plaintext.

	Attack	#Rounds	Data	Time	Memory	Reference
T-TWINE-80	Imp. diff	25	$2^{65.5}$ CTP	$2^{70.86}$	2^{66}	[22]
	Integral	26	$2^{70.58}$ CTP	$2^{72.62}$	$2^{67.62}$	Sect. 4.1
		27	$2^{70.95}$ CTP	$2^{75.79}$	$2^{71.08}$	Sect. 5.1
T-TWINE-128	Imp. diff	27	2^{64} CTP	$2^{120.83}$	2^{118}	[22]
	Integral	27	$2^{71.58}$ CTP	$2^{109.54}$	$2^{90.58}$	Sect. 4.2
		28	$2^{72.27}$ CTP	$2^{113.38}$	$2^{94.32}$	Sect. 5.1

Outline. The rest of this paper is organized as follows. In Sect. 2, we briefly revisit the specifications of T-TWINE and the integral cryptanalysis using the bit-based division property. The detailed integral distinguishing attacks against T-TWINE is explained in Sect. 3. In Sect. 4, we describe the key recovery attacks against 26 and 27 rounds of T-TWINE-80 and T-TWINE-128, respectively. Then, the details of our attacks against 27 and 28 rounds of T-TWINE-80 and T-TWINE-128 using dynamically chosen plaintexts are presented in Sect. 5. Finally, the paper is concluded in Sect. 6.

2 Preliminaries

2.1 T-TWINE Specifications

The following notation is used throughout the rest of the paper:

- K : The 80 or 128 bits master key.
- K_j : The j^{th} nibble of K . The indices of the nibbles begin from 0.
- RK^i : The 32-bit round key used in round i .
- RK_j^i : The j^{th} nibble of RK^i . The indices of the nibbles begin from 0.
- T : The 64-bit tweak.
- T_j : The j^{th} nibble of the tweak T .
- RT^i : The 24-bit round tweak used in round i , where $RT^i \leftarrow t_0^i || t_1^i || t_2^i || t_3^i || t_4^i || t_5^i$, and t_j^i is the j^{th} nibble of RT^i .
- X^i : The 16 nibbles input to round i . The indices of the round begin from 1.
- X_j^i : j^{th} nibble of X^i .
- $x[m]$: m^{th} bit of the nibble x where $x[0]$ is the least significant bit.
- \oplus : The XOR operation.
- $||$: The concatenation operation.
- $Rotz(x)$: The z -bit left cyclic shift of x .

As we mentioned above, T-TWINE is an extension of the conventional block cipher TWINE. It takes a tweak of 64 bits as an extra input in addition to a block of plaintext with 64 bits in order to produce a block of ciphertext using 80 or 128 bits of a secret key. T-TWINE structure consists of three parts: data

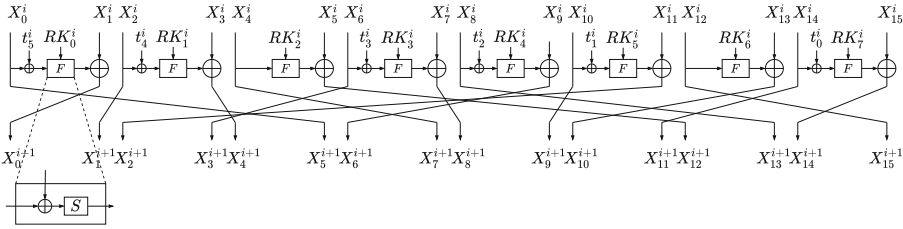


Fig. 1. T-TWINE round function

Table 2. Nibble shuffle π

h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[h]$	5	0	1	4	7	12	3	8	13	6	9	2	15	10	11	14
$\pi^{-1}[h]$	1	2	11	6	3	0	9	4	7	10	13	14	5	8	15	12

processing which is a slightly modified version of the equivalent part in TWINE to deal with the extra input, key scheduling function of TWINE, and tweak scheduling function. The two variants of T-TWINE are the same except in the key scheduling function.

Data Processing. The round function is based on a variant of Type-2 GFS [19] with 16 4-bit nibbles as depicted in Fig. 1. It consists of a nonlinear layer (F -function operations), round tweak XOR, and a diffusion layer which is a 16-nibble shuffle operation (π , see Table 2). The F -function operation is a round-key XOR followed by 4-bit Sbox (S , see Table 3). This round function is iterated 36 times in both variants where the diffusion layer is omitted from the last round.

Key Scheduling Function. Each variant of T-TWINE has its own key schedule. The key scheduling function is used to stretch 80/128 bits of the master key K to 36 32-bit round keys RK^i where $1 \leq i \leq 36$. For more details, see [15, 20].

Tweak Scheduling Function. A 64-bit tweak T is used to generate 36 24-bit round tweaks RT^i where $1 \leq i \leq 36$ using a permutation-based function. Firstly, the 64-bit tweak T is loaded to 16 4-bit nibbles t_j^i where $0 \leq j \leq 15$. In i -th round, the first 6 nibbles (t_0^i, \dots, t_5^i) are used as the round tweak RT^i , then these nibbles are shuffled using a 6-nibble permutation π^t , s.t. $(0, 1, 2, 3, 4, 5) \rightarrow (1, 0, 4, 2, 3, 5)$. After that, all nibbles are shifted by 6 nibbles to construct t_j^{i+1} where $0 \leq j \leq 15$ as depicted in Fig. 2.

2.2 Integral Cryptanalysis

Integral cryptanalysis was firstly introduced by Daemen *et al.* in [4] to analyze the security of the block cipher SQUARE. Subsequently, Knudsen and Wagner

Table 3. 4-bit Sbox (S) of T-TWINE in hexadecimal form

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	c	0	f	a	2	b	9	5	8	3	d	7	1	e	6	4

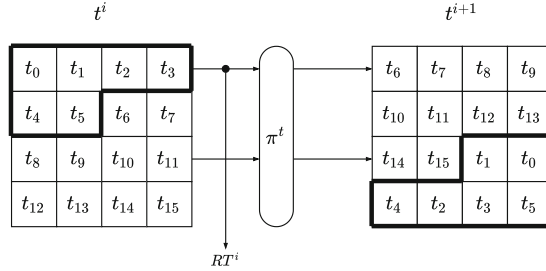


Fig. 2. Tweak schedule of T-TWINE

[10] formalized this technique. It is a chosen-plaintext attack and can be performed as follows. Firstly, the cryptanalyst constructs a set of plaintexts that has a constant value at some bits while the other bits vary through all possible values. After that, the cryptanalyst calculates the XOR sum of all bits (or some of them) on the corresponding ciphertext. If it is always 0 irrespective of the used secret key, these bits are called balanced. This property can be used to distinguish the block cipher under test from a random permutation.

Bit-Based Division Property. In [21], Todo and Morii proposed the *bit-based division property*, which can be used to build a longer integral distinguisher for block ciphers with block size less than 32 bits. Xiang *et al.* [23] overcome the problem of the restriction on the block size using *the division trails*. They proposed systematic rules to represent the bit-based division property propagation as a set of Mixed Integer Linear Programming (MILP) constraints. Hence, we can use MILP solvers to search for a distinguisher.

Definition 1 (Bit-based Division Property [21]). Let \mathbb{X} be a multiset whose elements take a value of \mathbb{F}_2^n . When the multiset \mathbb{X} has the division property $\mathcal{D}_{\mathbb{K}}^{1^n}$, where \mathbb{K} denotes a set of n -dimensional vectors whose i -th element takes 0 or 1, it fulfills the following conditions:

$$\bigoplus_{x \in \mathbb{X}} x^u = \begin{cases} \text{unknown} & \text{if there exists } \mathbf{k} \in \mathbb{K} \text{ s.t. } \mathbf{u} \succeq \mathbf{k}, \\ 0 & \text{otherwise.} \end{cases}$$

where $x^u = \prod_{i=1}^n x[i]^{u[i]}$, $\mathbf{u} \succeq \mathbf{k}$ if $u[i] \geq k[i] \forall i$, and $x[i]$, $u[i]$ are the i -th bits of x and u , respectively.

Definition 2 (Division Trail [23]). Let f_r denote the round function of an iterated block cipher. Assume that the input multiset to the block cipher has the

initial division property $\mathcal{D}_{\{\mathbf{k}\}}^{1^n}$, and denote the division property after i -round propagation through f_r by $\mathcal{D}_{\mathbb{K}_i}^{1^n}$. Thus, we have the following chain of division property propagations: $\{\mathbf{k}\} \stackrel{\text{def}}{=} \mathbb{K}_0 \xrightarrow{f_r} \mathbb{K}_1 \xrightarrow{f_r} \mathbb{K}_2 \xrightarrow{f_r} \dots \xrightarrow{f_r} \mathbb{K}_r$. Moreover, for any vector $\mathbf{k}_i^* \in \mathbb{K}_i (i \geq 1)$, there must exist a vector $\mathbf{k}_{i-1}^* \in \mathbb{K}_{i-1}$ such that \mathbf{k}_{i-1}^* can propagate to \mathbf{k}_i^* by the division property propagation rules. Furthermore, for $(\mathbf{k}_0, \mathbf{k}_1, \dots, \mathbf{k}_r) \in \mathbb{K}_0 \times \mathbb{K}_1 \times \dots \times \mathbb{K}_r$, if \mathbf{k}_{i-1} can propagate to \mathbf{k}_i for all $i \in \{1, 2, \dots, r\}$, we call $(\mathbf{k}_0, \mathbf{k}_1, \dots, \mathbf{k}_r)$ an r -round division trail.

Using the division trial, the search process for an integral distinguisher is converted to check if the division trail $\mathbf{k}_0 \rightarrow \dots \rightarrow e_i$ (a unit vector whose i -th element is 1) does exist or not. If it does not exist, then the i -th bit of r -round output is balanced. This process can be modeled efficiently as an MILP optimization problem. Further details can be found in [5, 17, 23].

In the following, we summarize the MILP models of the propagation rules of the bit-based division property through the basic operations in block ciphers.

- Model for COPY: Let $(a) \xrightarrow{COPY} (b_1, b_2, \dots, b_m)$ denote the division trail through COPY function, where a single bit (a) is copied to m bits. Then, it can be described using the following MILP constraints:

$$a - b_1 - b_2 - \dots - b_m = 0, \text{ where } a, b_1, b_2, \dots, b_m \text{ are binary variables.}$$

- Model for XOR: Let $(a_1, a_2, \dots, a_m) \xrightarrow{XOR} (b)$ denote the division trail through an XOR function, where m bits are compressed to a single bit (b) using an XOR operation. Then, it can be described using the following MILP constraints:

$$a_1 + a_2 + \dots + a_m - b = 0, \text{ where } a_1, a_2, \dots, a_m, b \text{ are binary variables.}$$

- Model for S-boxes: The division property through an S-box can be obtained by representing the S-Box using its algebraic normal form (ANF) [23]. The division trail though an n -bit S-box can be represented as a set of $2n$ -dimensional binary vectors $\in \{0, 1\}^{2n}$ which has a convex hull. The H-Representation of this convex hull can be computed using readily available functions such as `inequality_generator()` function in SageMath¹ which returns a set of linear inequalities that describe these vectors. We use this set of inequalities as MILP constraints to present the division trail though the S-box.

3 Integral Distinguishing Attacks

Since T-TWINE is an extension of TWINE which has 16-round integral distinguisher using 2^{63} chosen plaintexts [24], in this section we study the effect of the freedom gained by adding a tweak to the structure. Thereby, we report the

¹ <http://www.sagemath.org/>.

result regarding the integral distinguishers in the three attack settings: chosen tweak, chosen tweak-plaintext, and chosen tweak-ciphertext. To this end, we utilize MILP models of the propagation rules of the bit-based division property described in the previous section to automate the search process using Gurobi optimizer [8]. We obtain the best distinguisher in two steps. In the first step, we look for a distinguisher that covers the maximum number of rounds irrespective of the data complexity. Then, we try to reduce the data complexity of the longest one in the second step. We use the following notation to present the status of each nibble of the tweak, plaintext, and ciphertext:

- | |
|---|
| C |
|---|

 each bit of the nibble is fixed to constant.
- | |
|---|
| A |
|---|

 all bits of the nibble are active.
- | |
|---|
| A |
|---|

 all bits of the nibble are active except one arbitrary bit is constant.
- | |
|---|
| B |
|---|

 each bit of the nibble is balanced (the XOR sum is zero).
- | |
|---|
| U |
|---|

 a nibble with unknown status.

Chosen Tweak Setting. In this setting, all the plaintext bits are fixed to constant values and some or all the bits of the tweak are active while the remaining bits are constant.

In the first step, we set all bits of the tweak to active. We then target r rounds and use our MILP model to search for some balanced bits. If there is at least one balanced bit, we increase the target rounds to $r + 1$ and repeat the search process in the same way. Otherwise, we conclude that the distinguisher with the maximum number of rounds based on our model covers r rounds. Based on our evaluation, there is no distinguisher for 12 or more rounds and the longest distinguisher is an 11-round one. In the second step, we try to reduce the data complexity of that 11-round distinguisher by minimizing the number of active nibbles in the tweak. To this end, we start with only one active nibble and if there is no balanced bits, we progressively increase the number of active nibbles. Fortunately, we find two distinguishers with only one active nibble as shown below:

Plaintext	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
Tweak	c	c	c	c	c	c	c	c	c	c	c	c	c	A	c
Tweak	c	c	c	c	c	c	c	c	c	c	c	c	c	c	A

$\xrightarrow{11R}$

u	u	u	u	u	u	u	u	u	u	u	u	u	u	B	u	u	u	u	u	u	u	u	u	u	u	u
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$\xrightarrow{11R}$

u	u	u	u	u	u	u	u	u	u	u	u	u	u	B	u	u	u	u	u	u	u	u	u	u	u	u
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

It should be mentioned that the designers have reported in [15] a different 11-round integral distinguisher in which the plaintext nibbles are fixed to constant, the three nibbles (5, 10, 11) in the tweak are actives, and the remaining nibbles in the tweak are fixed to constant. This distinguisher has two balanced nibbles (0, 11) in the ciphertext side as shown below. However, when we test this distinguisher using our MILP model with the same input settings, we confirmed that there is only one balanced nibble (11) in the ciphertext side.

Plaintext	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	
Tweak	c	c	c	c	c	A	c	c	c	c	A	A	c	c	c	c	$\xrightarrow{11R},$ B u u u u u u u u u u u u B u u u u ✘ (15)			
Tweak	c	c	c	c	c	A	c	c	c	c	A	A	c	c	c	c	$\xrightarrow{11R},$ u u u u u u u u u u u u B u u u u ✔ (Ours)			

Since the data complexity for each one of the two 11-round integral distinguishers we have proposed is 2^4 , we have verified the correctness of them experimentally to validate our results. Additionally, the data complexity of the 11-round distinguisher with the same input settings as the distinguisher reported in [15] is 2^{12} , we also have verified experimentally that it has only one balanced nibble (11) in the ciphertext side which is consistent with the result using our MILP model².

Chosen Tweak-Plaintext Setting. In this setting, some of plaintext bits are active and the remaining bits are constant. For the tweak, some or all bits are active and the remaining bits are constant.

Since the goal of the first step is to obtain the longest distinguisher, we set the 64 bits of the tweak and 63 bits of the plaintext to active and the remaining bit of the plaintext to constant³. We then target r rounds and iterate over the 64 positions of the constant bit until we find some balanced bits or terminate without finding any. In the first case, we increase the target rounds to $r + 1$ and repeat the search process in the same way. Otherwise, we conclude that the distinguisher with the maximum number of rounds based on our model covers r rounds. In our evaluation, we found that the 19-round distinguisher is the longest one.

In order to convert the distinguishing attack to a key recovery attack applicable for both variants T-TWINE-80 and T-TWINE-128, the data complexity of the distinguisher must be less than 2^{80} . Therefore, we limit the search process to find a distinguisher that needs up to 80 active bits.

During the second step, we try to reduce the data complexity by minimizing the number of active bits in both plaintext and tweak. We follow the technique described in [18] to reduce the active bits of the plaintext. In particular, we repeat the previous step for 19 rounds and instead of stopping the search process if there are some balanced bits, we keep a record of the position of the constant bit in case of no balanced bits. In our evaluation, there are 32 bits corresponding to the nibbles (1, 3, 5, 7, 9, 11, 13, 15) that must be active to obtain 19-round distinguisher and the remaining bits may be active or constant. After that, we try all the combinations of 2 out of 32 bits that might be constant and check if the 19-round distinguisher exists. Unfortunately, such distinguisher does not exist if we set any two bits in the plaintext to constant. Regarding the active bits reduction in the tweak, we start with only one active nibble and if there is no distinguisher, we progressively increase the number of active nibbles.

² The code can be found at: <https://github.com/mhgharieb/Integral-Attack-T-TWINE>.
³ The data complexity of plaintext must be less than the full codebook because using the full codebook of any permutation (a random permutation or a block cipher) always gives a balanced output.

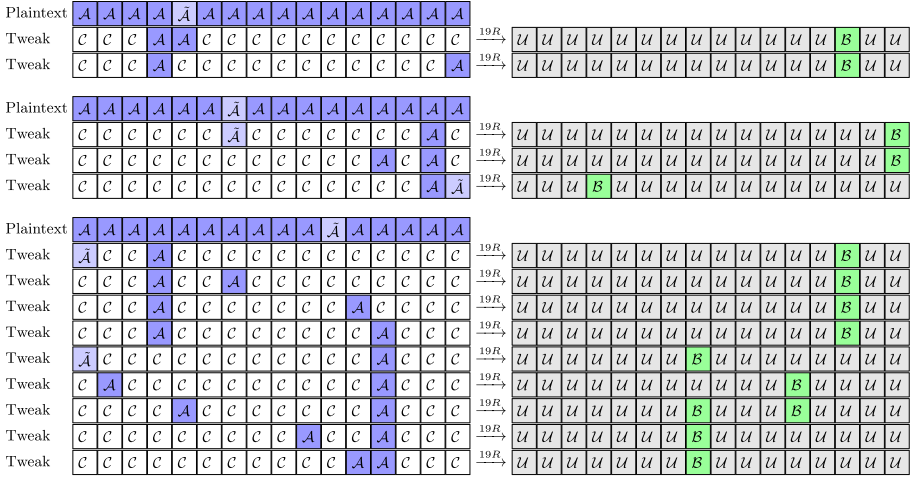


Fig. 3. 104 19-round integral distinguishers in chosen tweak-plaintext setting, where the three groups consist of $4 \times (1 + 1) = 8$, $4 \times (4 + 1 + 4) = 36$, and $4 \times (4 + 1 + 1 + 1 + 4 + 1 + 1 + 1 + 1) = 60$ distinguishers.

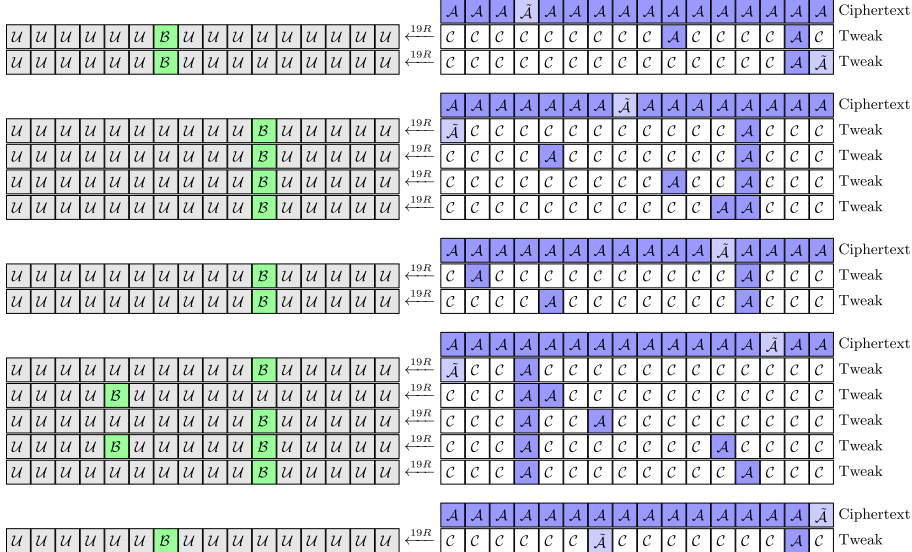


Fig. 4. 104 19-round Integral distinguishers in chosen tweak-ciphertext setting, where the five groups consist of 20, 28, 8, 32, and 16 distinguishers.

In our evaluation, there are several 19-round integral distinguishers using tweak with two active nibbles. Moreover, we are able to reduce the active bits to 7 bits for some of them and 6 bits for the distinguisher that we will use during the

key recovery attacks. Figure 3 summarizes 40 distinguishers with $2^8 \times 2^{63} = 2^{71}$, and 64 distinguishers with $2^7 \times 2^{63} = 2^{70}$ chosen tweak-plaintext combinations.

Chosen Tweak-Ciphertext Setting. In this setting, some of ciphertext bits are active and the remaining bits are constant. For the tweak, some or all bits are active and the remaining bits are constant.

We followed the same technique we have used in chosen tweak-plaintext setting and we found that the 19-round integral distinguisher is the longest one. Like chosen tweak-plaintext setting, the distinguisher does not exist if there are two constant bits in the ciphertext. Also, there are several two active nibbles combinations of the tweak that lead to 19-round distinguisher. Moreover, we are able to reduce, for some of them, the active bits to only 7. Figure 4 summarizes 104 19-round integral distinguishers, 64 of them need $2^7 \times 2^{63} = 2^{70}$ chosen tweak-ciphertext combinations and the remaining need $2^8 \times 2^{63} = 2^{71}$ chosen tweak-ciphertext combinations.

4 Integral Attacks on T-TWINE

We convert the distinguishing attacks described in the previous section to key recovery attacks against reduced-round versions of T-TWINE. In particular, we target 26 and 27 rounds of T-TWINE-80 and T-TWINE-128, respectively, using the following 19-round distinguisher that needs 6 and 63 active bits of the tweak and the plaintext, respectively:

$$\begin{array}{l} \text{Plaintext : } (\mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}_3, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}) \\ \text{Tweak : } (\mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{A}_{1,3}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{A}, \mathcal{C}) \\ \quad \quad \quad \downarrow 19R \\ (\mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{U}, \mathcal{B}) \end{array}$$

where \mathcal{A}_3 means all bits of the nibble are active except bit 3, counted from the least significant bit, is constant and $\mathcal{A}_{1,3}$ means bits (0 and 2) are active and bits (1 and 3) are constant.

In the following, we revisit the Meet-in-the-Middle technique [16] and Partial-Sum technique [7] that we use to enhance the time complexities of our proposed attacks.

Meet-in-the-Middle Technique. Let Z_j^i , ($0 \leq j \leq 7$) denote the output of the F functions in i -th round of T-TWINE. Consider the 19-round distinguisher mentioned above, then the nibble X_{15}^{20} is balanced ($\bigoplus X_{15}^{20} = 0$). Since this nibble can be expressed as a linear combination of Z_7^{20} and X_{14}^{21} , we can obtain the following relation

$$\bigoplus Z_7^{20} = \bigoplus X_{14}^{21}$$

In meet-in-the-middle technique [16], each sum is independently computed from ciphertexts (*e.g.*, see Fig. 5) and the subkeys used during the computation

are stored in two different tables indexed by the value of the sum. After that, we consider the matches between the two tables, in the same manner of the meet-in-the-middle attack, as candidate subkeys because they satisfy the previous relation. Since the procedure to obtain both $\bigoplus Z_7^{20}$ and $\bigoplus X_{14}^{21}$ independently involves less number of subkeys than the one to obtain $\bigoplus X_{15}^{20}$ directly, the time complexity will be improved.

Partial-Sum Technique. Ferguson *et al.* introduced the partial-sum technique to improve the time complexity of integral attacks [7]. Suppose the key recovery procedure during the integral cryptanalysis involves N operations, κ -bit subkey and $2^{|I|}$ ciphertexts, then the time complexity of the direct computation will be $N \times 2^{|I|+\kappa}$ operations. Using the partial-sum technique, this time complexity can be improved as follows. We firstly store the ciphertexts that appear odd times in the memory whereas the ciphertexts that appear even times are discarded since they have no effect on the balanced property. Then, we guess a part of the subkey (κ_1 -bit) and partially decrypt the ciphertexts through a single operation to an intermediate state with $|I_1|$ -bit size (that can have up to $2^{|I_1|}$ values) such that $|I_1| \leq |I|$. The time complexity of this step is $2^{|I|+\kappa_1}$ operations. After that, we repeat the step of storing the values that appear odd times and partially decrypting the intermediate state using κ_i -bit to get another intermediate state with $|I_i|$ -bit size such that $|I_i| \leq |I_{i-1}|$. The time complexity of the i -th step will be $2^{|I_{i-1}|+\kappa_1+\dots+\kappa_i}$ where I_0 is I , and the whole time complexity will be

$$\sum_{i=1}^N 2^{|I_{i-1}|+\kappa_1+\dots+\kappa_i} < \sum_{i=1}^N 2^{|I|+\kappa} = N \times 2^{|I|+\kappa}$$

In the following, we give the details of the key recovery attack against T-TWINE-80.

4.1 Attack on 26-Round T-TWINE-80

The ciphertexts of 26-round of T-TWINE-80 can be written as X^{27} . The process of obtaining $\bigoplus X_{15}^{20}$ involves the following 27 round keys (See Fig. 5):

$$RK^{26}, RK_{[0,1,2,3,4,5,7]}^{25}, RK_{[0,1,2,6,7]}^{24}, RK_{[0,4,6]}^{23}, RK_{[4,5]}^{22}, RK_5^{21}, RK_7^{20}$$

However, we only need to guess 76 bits in 19 round keys and the other 8 round keys can be computed based on the key schedule as follows:

$$\begin{aligned} RK_0^{24} &= RK_7^{25} \oplus S(RK_6^{26} \oplus (0||CON_L^{26})), & RK_1^{24} &= RK_5^{26}, \\ RK_2^{24} &= S^{-1}(RK_7^{26} \oplus RK_0^{25}) \oplus S(RK_7^{24}), & RK_4^{23} &= RK_0^{26}, \\ RK_6^{23} &= RK_1^{26} \oplus (0||CON_H^{26}), & RK_4^{22} &= RK_0^{25}, \\ RK_5^{21} &= RK_4^{26}, & RK_7^{20} &= RK_6^{26} \oplus S(RK_2^{26}) \oplus (0||CON_L^{26}). \end{aligned}$$

where CON_L^{26} and CON_H^{26} are predefined constants.

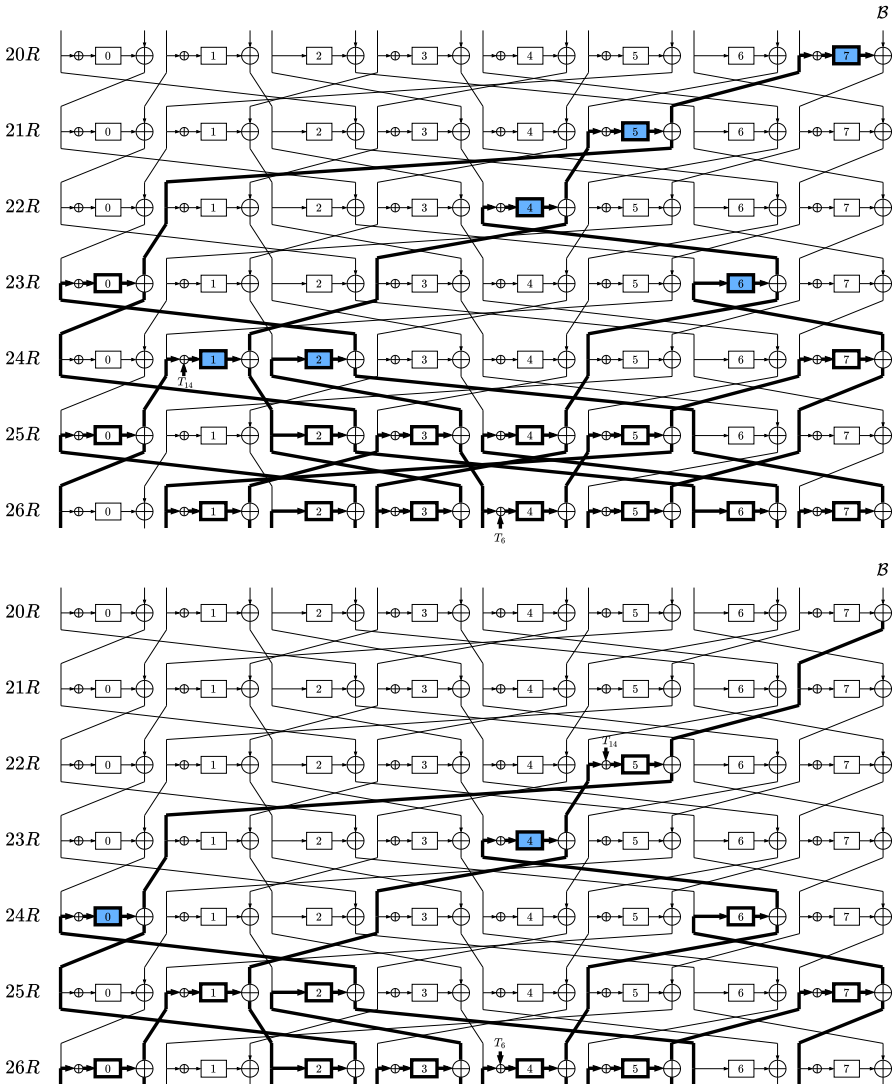


Fig. 5. Analysis rounds of T-TWINE-80 where the upper part is used during computing $\oplus Z_7^{20}$ and the lower part is used during computing $\oplus X_{14}^{21}$.

Key Recovery Procedure. We firstly construct a data structure where all the bits of the plaintext X^1 are active except the bit $X_6^1[3]$ which is fixed to constant. For the tweak, the 6 bits $T_6[0, 2] || T_{14}$ are active whereas the other bits are fixed to constant. We then ask the encryption oracle to obtain the corresponding ciphertext (X^{27}). After that, we initialize two empty hash tables H_Z and H_X with 2^{56} and 2^{40} entries to store the values of $\oplus Z_7^{20}$ and $\oplus X_{14}^{21}$, respectively, indexed by the round keys used during the computations.

Since obtaining $\bigoplus Z_7^{20}$ (the upper part of Fig. 5) requires much more computation than obtaining $\bigoplus X_{14}^{21}$ (the lower part of Fig. 5), we only explain the procedure to obtain $\bigoplus Z_7^{20}$. The attack starts by storing the values of $X_{[0,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}^{27} || T_6[0, 2] || T_{14}$ that appear odd times in a list called the state S_0 which has a size of up to 2^{66} 66-bit values. Then, we guess at the i -th step a round key (or deduce it based on the key schedule as shown above) and partially decrypt the values in the state S_{i-1} , then store the values of the output that appear odd times in a new state S_i . For example, we guess at step 1 RK_2^{26} and partially decrypt X_4^{27} and X_5^{27} to obtain $X_5^{26} = X_5^{27} \oplus S(X_4^{27} \oplus K_2^{26})$. The state size after compression is up to 2^{62} 62-bit values. The time complexity of this step is $2^4 \times 2^{66} = 2^{70}$ F -function operations. Table 4 summarizes the steps of the attack procedure.

Finally, we access the hash tables (H_Z, H_X) for each 76-bit key, and we consider a 76-bit key as a candidate if the two entries are equal. The 4 balanced bits lead to 4 bits filtration, therefore we get 2^{72} 76-bit candidates for the round keys when we use a single data structure. We can reduce the number of the candidates by repeating the attack using another data structure. Thanks to the key schedule, we can obtain 2^{76} 80-bit candidates for the master key corresponding to these 2^{72} 76-bit round keys by guessing 4-bit round key. The details of this step can be found in Appendix A. We then get the right master key by exhaustively searching over these candidates using 2 plaintext/ciphertext pairs.

Attack Complexity. When we use a single data structure, we need $2^6 \times 2^{63} = 2^{69}$ queries to the encryption oracle. From Table 4, we need $2^{78.13}$ F -function operations to compute $\bigoplus Z_7^{20}$. Using the same method, we need $2^{59.91}$ F -Function operations to compute $\bigoplus X_{14}^{21}$.

We then access the hash tables (H_Z, H_X) sequentially to retrieve 2 4-bit words. For simplicity, we consider the time to retrieve a single 4-bit word as a one F -function operation. Therefore, for this step, we need $2^{56} \times (1 + 2^{20}) \approx 2^{76}$ F -function operations. Consequently, we got 2^{72} 76-bit candidates of the round keys. As shown in Appendix A, we need 145 F -function operations for each candidate to get the corresponding 2^4 80-bit candidates of the master key. The exhaustive search over the candidates to get the right master key takes $2^{76} + 2^{12}$ 26-round encryptions. Therefore, the total time complexity is $2^{69} + 1 \times \frac{2^{78.13} + 2^{59.91}}{8 \times 26} + \frac{2^{76}}{8 \times 26} + \frac{145 \times 2^{72}}{8 \times 26} + 2^{76} + 2^{12} \approx 2^{76.11}$ 26-round encryptions. The memory complexity is dominated by storing the part of the ciphertexts involved during the computation of $\bigoplus Z_7^{20}$ (the state S_0) which is 2^{66} 66-bit blocks that is equivalent to $2^{66.04}$ 64-bit blocks. As shown in Table 5, the lowest time complexity can be achieved using 3 data structures and in this case the data, time, and memory complexities are $3 \times 2^6 \times 2^{63} = 2^{70.58}$ chosen tweak-plaintext combinations, $2^{72.62}$ 26-round encryptions, and $2^{67.62}$ 64-bit blocks, respectively.

Table 4. Summary of the procedure to obtain $\bigoplus Z_7^{20}$ where 'Size' refers to the size of the intermediate state S_i after the partial decryption at each step, the nibbles X_j^r in the state S_{i-1} are replaced by the nibbles X_j^r s in the state S_i during the i -th step, and 'Complexity' is measured in term of F -function operations except step 0 is measured in number of memory accesses (MA).

Step	Key	Size	The State (S_i)	Complexity
0	-	2^{66}	$X_0^{27}, X_2^{27}, X_3^{27}, X_4^{27}, X_5^{27}, X_6^{27}, X_7^{27}, X_8^{27}, X_9^{27}, X_{10}^{27}, X_{11}^{27}, X_{12}^{27}, X_{13}^{27}, X_{14}^{27}, X_{15}^{27}, T_6, T_{14}$	$2^{66} MA$
1	RK_2^{26}	2^{62}	$X_0^{27}, X_2^{27}, X_3^{27}, X_5^{26}, X_6^{27}, X_7^{27}, X_8^{27}, X_9^{27}, X_{10}^{27}, X_{11}^{27}, X_{12}^{27}, X_{13}^{27}, X_{14}^{27}, X_{15}^{27}, T_6, T_{14}$	$2^4 \times 2^{66} = 2^{70}$
2	RK_5^{26}	2^{58}	$X_0^{27}, X_2^{27}, X_3^{27}, X_5^{26}, X_6^{27}, X_7^{27}, X_8^{27}, X_9^{27}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{14}^{27}, X_{15}^{27}, T_6, T_{14}$	$2^8 \times 2^{62} = 2^{70}$
3	RK_7^{26}	2^{54}	$X_0^{27}, X_2^{27}, X_3^{27}, X_5^{26}, X_6^{27}, X_7^{27}, X_8^{27}, X_9^{27}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{14}^{26}, T_6, T_{14}$	$2^{12} \times 2^{58} = 2^{70}$
4	RK_0^{25}	2^{50}	$X_1^{25}, X_2^{27}, X_3^{27}, X_5^{27}, X_6^{27}, X_7^{27}, X_8^{27}, X_9^{27}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}, T_6, T_{14}$	$2^{16} \times 2^{54} = 2^{70}$
5	RK_3^{26}	2^{50}	$X_1^{25}, X_2^{27}, X_3^{27}, X_5^{26}, X_7^{26}, X_8^{27}, X_9^{27}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}, T_6, T_{14}$	$2^{20} \times 2^{50} = 2^{70}$
6	RK_1^{24}	2^{46}	$X_3^{24}, X_2^{27}, X_3^{27}, X_5^{26}, X_7^{26}, X_8^{27}, X_9^{27}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}, T_6$	$2^{20} \times 2^{50} = 2^{70}$
7	RK_4^{26}	2^{44}	$X_3^{24}, X_2^{27}, X_3^{27}, X_5^{26}, X_7^{26}, X_8^{26}, X_9^{26}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}$	$2^{24} \times 2^{46} = 2^{70}$
8	RK_1^{26}	2^{44}	$X_3^{24}, X_5^{26}, X_3^{26}, X_7^{26}, X_8^{26}, X_9^{26}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}$	$2^{28} \times 2^{44} = 2^{72}$
9	RK_3^{25}	2^{40}	$X_3^{24}, X_5^{26}, X_6^{26}, X_7^{26}, X_8^{25}, X_9^{26}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}$	$2^{32} \times 2^{44} = 2^{76}$
10	RK_5^{25}	2^{36}	$X_3^{24}, X_5^{26}, X_6^{26}, X_7^{25}, X_{11}^{25}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}$	$2^{36} \times 2^{40} = 2^{76}$
11	RK_7^{24}	2^{32}	$X_3^{24}, X_5^{26}, X_6^{26}, X_7^{25}, X_{12}^{25}, X_{13}^{27}, X_{15}^{24}, X_{15}^{26}$	$2^{40} \times 2^{36} = 2^{76}$
12	RK_2^{24}	2^{28}	$X_3^{24}, X_5^{24}, X_6^{26}, X_7^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{24}$	$2^{40} \times 2^{32} = 2^{72}$
13	RK_6^{26}	2^{28}	$X_3^{24}, X_5^{24}, X_6^{26}, X_7^{26}, X_{12}^{26}, X_{13}^{26}, X_{15}^{24}$	$2^{44} \times 2^{28} = 2^{72}$
14	RK_4^{24}	2^{24}	$X_3^{24}, X_5^{24}, X_7^{26}, X_9^{25}, X_{12}^{26}, X_{15}^{24}$	$2^{48} \times 2^{28} = 2^{76}$
15	RK_6^{23}	2^{20}	$X_3^{24}, X_5^{24}, X_7^{26}, X_{12}^{26}, X_{13}^{23}$	$2^{48} \times 2^{24} = 2^{72}$
16	RK_1^{22}	2^{16}	$X_5^{24}, X_7^{26}, X_9^{22}, X_{12}^{26}$	$2^{48} \times 2^{20} = 2^{68}$
17	RK_2^{25}	2^{12}	$X_5^{24}, X_9^{25}, X_9^{22}$	$2^{52} \times 2^{16} = 2^{68}$
18	RK_0^{23}	2^8	X_1^{23}, X_9^{22}	$2^{56} \times 2^{12} = 2^{68}$
19	RK_5^{21}	2^4	X_{11}^{21}	$2^{56} \times 2^8 = 2^{64}$
20	RK_7^{20}	1	$\bigoplus Z_7^{20} = \bigoplus S(X_{11}^{21} \oplus RK_7^{20})$	$2^{56} \times 2^4 = 2^{60}$

Table 5. The data, time, and memory complexities using multiple data structures.

	Data	Time Complexity	Memory
1	2^{69}	$2^{69} + 1 \times \frac{2^{78.13} + 2^{59.91}}{8 \times 26} + \frac{2^{76}}{8 \times 26} + \frac{145 \times 2^{72}}{8 \times 26} + 2^{76} + 2^{12} \approx 2^{76.11}$	$2^{66.04}$
2	2^{70}	$2^{70} + 2 \times \frac{2^{78.13} + 2^{59.91}}{8 \times 26} + \frac{2^{76} + 2^{72}}{8 \times 26} + \frac{145 \times 2^{68}}{8 \times 26} + 2^{72} + 2^8 \approx 2^{73.03}$	$2^{67.04}$
3	$2^{70.58}$	$2^{70.58} + 3 \times \frac{2^{78.13} + 2^{59.91}}{8 \times 26} + \frac{2^{76} + 2^{72} + 2^{68}}{8 \times 26} + \frac{145 \times 2^{64}}{8 \times 26} + 2^{68} + 2^4 \approx 2^{72.62}$	$2^{67.62}$
4	2^{71}	$2^{71} + 4 \times \frac{2^{78.13} + 2^{59.91}}{8 \times 26} + \frac{2^{76} + 2^{72} + 2^{68} + 2^{64}}{8 \times 26} + \frac{145 \times 2^{60}}{8 \times 26} + 2^{64} \approx 2^{72.95}$	$2^{68.04}$

4.2 Attack on 27-Round T-TWINE-128

The ciphertexts of 27-round of T-TWINE-128 can be written as X^{28} . The process of obtaining $\bigoplus X_{15}^{20}$ involves the following 35 round keys:

$$RK^{27}, RK^{26}, RK_{[0,1,2,3,4,5,7]}^{25}, RK_{[0,1,2,6,7]}^{24}, RK_{[0,4,6]}^{23}, RK_{[4,5]}^{22}, RK_5^{21}, RK_7^{20}$$

However, we only need to guess 116 bits in 29 round keys and the other 6 round keys can be computed based on the key schedule as follows:

$$\begin{aligned} RK_0^{23} &= RK_4^{27}, & RK_6^{23} &= RK_2^{27}, \\ RK_4^{22} &= RK_6^{27} \oplus S(RK_7^{27}), & RK_5^{22} &= RK_0^{26}, \\ RK_5^{21} &= RK_0^{25}, & RK_7^{20} &= RK_1^{27} \oplus S(RK_5^{25}) \oplus (0||CON_L^{24}) \oplus (0||CON_H^{27}). \end{aligned}$$

where CON_L^{24} and CON_H^{27} are predefined constants.

Key Recovery Procedure. Using the same procedure we have applied in the previous section, we can recover 2^{112} 116-bit candidates of the round keys and then retrieve 2^{124} 128-bit candidates of the master key by guessing 12 bits. The number of the candidates can be reduced by repeating the attack several times using different values of the constant bits in the data structure.

Attack Complexity. When we use a single data structure, we need approximately $2^{113.83}$ F -function operations to fill the hash tables, then we need additionally 2^{116} F -function operations to access the tables and recover 2^{112} 116-bit candidates. Thus, we retrieve the right master key using 2×2^{124} 27-round encryptions. By repeating the attack 6 times, we need $\frac{1}{8 \times 27} \times (6 \times 2^{113.83} + 2^{116} + 2^{112} + \dots + 2^{96}) + 2^{104} + 2^{40} = 2^{109.54}$ 27-round encryptions to retrieve the right master key. Hence, the data complexity is $6 \times 2^{69} = 2^{71.58}$ chosen tweak-plaintext combinations. The memory complexity is dominated by storing the values of $\bigoplus Z_7^{20}$ in the hash table H_Z . Therefore, we need 6×2^{92} 4-bit blocks which is equivalent to $2^{90.58}$ 64-bit blocks.

5 Attacking One More Round

Chu *et al.* [3] presented a general method to use the dynamically chosen plaintexts idea in order to attack one more round in the integral cryptanalysis by adding this round before the distinguisher. In general, appending rounds before the integral distinguisher may lead to use the full codebook of the plaintext. However, the dynamically chosen plaintext method guarantees that we will not use the full codebook of the plaintext. In this section, we explain how we can prepend one round before the integral distinguisher. Consequently, we can target 27 and 28 rounds of T-TWINE-80 and T-TWINE-128, respectively.

The core idea of the method is to express one of the constant bits (c) of the distinguisher input as a non-linear boolean function in some plaintext bits (x)

and key bits (k) *i.e.*, $c = f(x, k)$. Then, we guess the key bits (k) and carefully select a specific plaintext set \mathcal{D}_k^c that guarantees the constant bit c is fixed to 0 or 1 while the other bits satisfy the distinguisher input. Consequently, the whole plaintext set used during the attack will be $\bigcup \mathcal{D}_k^c$.

In our attack, the plaintext is X^1 and the distinguisher input is X^2 . Therefore, we have to select the plaintexts such that $X_6^2[3]$ (the most significant bit of X_6^2) is fixed to 0 or 1 while the other bits of X^2 are active. From T-TWINE structure, $X_6^2[3] = X_9^1[3] \oplus S(X_8^1 \oplus k)[3]$ where $k = RK_4^1 \oplus RT_2^1$.

Based on the algebraic normal form of T-TWINE's Sbox, $X_6^2[3]$ can be expressed as follows:

$$X_6^2[3] = X_9^1[3] \oplus 1 \oplus x[0] \oplus x[2] \oplus (x[0] \cdot x[1]) \oplus (x[1] \cdot x[2]) \oplus (x[0] \cdot x[1] \cdot x[2]) \\ \oplus (x[0] \cdot x[1] \cdot x[3]) \oplus (x[1] \cdot x[2] \cdot x[3])$$

where $x[i] = X_8^1[i] \oplus k[i]$ and $k[i] = RK_4^1[i] \oplus RT_2^1[i]$. Therefore, $X_6^2[3]$ depends on the 5 bits $X_8^1 || X_9^1[3]$ and the 4 bits of the round key RK_4^1 .

The procedure to determine the suitable plaintext set in our attack is as follows:

1. Initialize 32 empty lists namely \mathcal{D}_k^0 and \mathcal{D}_k^1 where $0 \leq k \leq 15$.
2. For each possible value of k and for all 2^5 possible values of $X_8^1 || X_9^1[3]$, compute $X_6^2[3]$ and store $X_8^1 || X_9^1[3]$ in \mathcal{D}_k^0 if $X_6^2[3]$ is 0 or in \mathcal{D}_k^1 if $X_6^2[3]$ is 1.
3. For each $C := \{c_k | 0 \leq k \leq 15\} \in \mathbb{F}_2^{16}$, if $|\bigcup_k \mathcal{D}_k^{c_k}| < 2^5$, export C and its corresponding $\{\mathcal{D}_k^{c_k}\}$ as a possible plaintext set.

Based on our evaluation, there are 32 plaintext sets of $\{\mathcal{D}_k^{c_k}\}$. In each set, there are 31 out of 32 possible values of $X_8^1 || X_9^1[3]$. To validate these sets, we perform an extra step as follows: for each k , we construct X_8^1, X_9^1 such that $X_8^1 || X_9^1[3] \in \mathcal{D}_k^{c_k}$ and $X_9^1[2] || X_9^1[1] || X_9^1[0]$ takes all possible values, then compute $X_6^2 = X_9^1 \oplus S(X_8^1 \oplus k)$, after that, we check if $X_8^1 || X_6^2[2] || X_6^2[1] || X_6^2[0]$ takes all 128 possible values and $X_6^2[3] = c_k$ or not. Table 6 in [6] depicts an example of these sets in which $X_8^1 || X_9^1[3]$ does not take the value of 000000.

Data Collection. We firstly construct a data structure in which all bits of X^1 take all the possible values except $X_8^1 || X_9^1[3] \in \bigcup \mathcal{D}_k^{c_k}$. For the tweak, all bits are fixed to constant except the 6 bits ($T_3, T_{12}[0, 2]$) take all the possible values. Then, we ask the encryption oracle to obtain the corresponding ciphertexts and store the ciphertext associated with the active bits of the tweak in a hash table indexed by the value of $X_8^1 || X_9^1[3]$. Therefore, the data complexity of a single structure is $2^6 \times (2^{64} - 2^{59}) \approx 2^{69.95}$ chosen tweak-plaintext combinations.

5.1 Key Recovery Attacks

T-TWINE-80. We firstly guess the value of RK_4^1 and based on the value of $k = RK_4^1 \oplus RT_2^1$, we select a set of 2^{69} ciphertexts corresponding to the plaintexts that include $\mathcal{D}_k^{c_k}$. After that, we apply the same steps described in Sect. 4.1 to

obtain 2^{72} candidates of the 76-bit round keys. It should be mentioned that the *relative* relations between the round keys involved in the analysis rounds are the same as in Sect. 4.1.

Using each value of RK_4^1 combined with 2^{72} 76-bit candidates of the round keys, we can compute 2^{72} 80-bit candidates of the master key. Subsequently, we get in total $2^4 \times 2^{72} = 2^{76}$ 80-bit candidates of the master. The right master key can be retrieved by the exhaustive search over these candidates using 2 pairs of plaintext/ciphertext.

The time complexity is $2^{69.95} + 2^4 \times (\frac{2^{78.13} + 2^{59.91}}{8 \times 27} + \frac{2^{76}}{8 \times 27} + \frac{145 \times 2^{72}}{8 \times 27}) + 2^{76} + 2^{12} \approx 2^{76.47}$ 27-round encryptions. The time complexity can be reduced to $2^{75.79}$ 27-round encryptions if we use two data structures ($2 \times 2^{69.95} = 2^{70.95}$ chosen tweak-plaintext combinations). The memory complexity is dominated by storing the ciphertexts associated with the active bits of the tweak. Therefore, the memory complexity will be $2^{71.08}$ 64-bit blocks.

T-TWINE-128. In the same manner, we can target 28 rounds of T-TWINE-128. By repeating the attack using different 5 data structures, we can retrieve the right master key. The data complexity is $5 \times 2^{69.95} = 2^{72.27}$ chosen tweak-plaintext combinations. The time complexity is $2^{113.38}$ 28-round encryptions. The memory complexity is $5 \times 2^4 \times 2^{92}$ 4-bit blocks which is equivalent to $2^{94.32}$ 64-bit blocks.

6 Conclusion

We studied the security of T-TWINE against the integral cryptanalysis. In particular, we showed that adding a tweak to the round function structure gives the attacker more room to target a large number of rounds in T-TWINE compared to TWINE. More precisely, we are able to construct several integral distinguishers that cover 19 rounds of T-TWINE whereas the longest distinguisher covers only 16 rounds of TWINE. Furthermore, we launched key recovery attacks against 27 and 28 of T-TWINE-80 and T-TWINE-128, respectively. Up to the authors' knowledge, the presented attacks are the best-published attacks against both variants of T-TWINE.

A Recovery of 80-bit keys of T-TWINE-80 attack

During the key recovery attack against T-TWINE-80, we have got 2^{72} 76-bit candidates of the 19 round keys $RK_{[0,1,2,3,4,5,6,7]}^{26}$, $RK_{[0,1,2,3,4,5,7]}^{25}$, $RK_{[6,7]}^{24}$, RK_0^{23} , RK_5^{22} as shown in Sect. 4.1. In this section, we describe how we can transform them to the 80-bit candidates of the master key.

Based on the key schedule of T-TWINE-80, these 19 round keys can be expressed as:

$$RK_0^{23} = V_1 \oplus CL_{10} \oplus CH_{13} \tag{1}$$

$$RK_0^{25} = V_2 \oplus CL_{12} \oplus CH_{15} \tag{2}$$

$$RK_0^{26} = V_3 \oplus CL_{13} \oplus CH_{16} \tag{3}$$

$$RK_4^{25} = V_4 \oplus CL_{15} \oplus CH_{18} \tag{4}$$

$$RK_4^{26} = V_5 \oplus CL_{16} \oplus CH_{19} \tag{5}$$

$$RK_5^{22} = V_6 \oplus CL_{17} \oplus CH_{20} \tag{6}$$

$$RK_3^{26} = V_7 \oplus CL_{19} \oplus CH_{22} \tag{7}$$

$$RK_3^{25} = V_8 \oplus CL_{18} \oplus CH_{21} \tag{8}$$

$$RK_5^{26} = V_9 \oplus CL_{21} \oplus CH_{24} \tag{9}$$

$$RK_1^{26} = V_{10} \oplus CL_{23} \oplus CH_{26} \tag{10}$$

$$RK_1^{25} = V_{11} \oplus CL_{22} \oplus CH_{25} \tag{11}$$

$$RK_2^{26} = V_{12} \tag{12}$$

$$RK_2^{25} = V_{13} \tag{13}$$

$$RK_5^{25} = V_{14} \oplus CL_{20} \oplus CH_{23} \tag{14}$$

$$RK_7^{24} = V_{15} \tag{15}$$

$$RK_7^{26} = V_2 \oplus CL_{12} \oplus CH_{15} \oplus S(V_{16} \oplus CL_7 \oplus CH_{10} \oplus S(V_{11}) \oplus S(V_{15})) \tag{16}$$

$$RK_6^{24} = V_{17} \oplus CL_4 \oplus CH_7 \oplus S(V_7) \oplus S(V_{16} \oplus CL_7 \oplus CH_{10} \oplus S(V_{11})) \oplus CL_{24} \tag{17}$$

$$RK_6^{26} = V_{18} \oplus CL_6 \oplus CH_9 \oplus S(V_9) \oplus S(V_{12}) \oplus CL_{26} \tag{18}$$

$$RK_7^{25} = V_{19} \oplus CL_{11} \oplus CH_{14} \oplus S(V_{18} \oplus CL_6 \oplus CH_9 \oplus S(V_9) \oplus S(V_{12})) \tag{19}$$

where $CL_i = 0 || CON_L^i$ and $CH_i = 0 || CON_H^i$ are predefined constants. The variables V_1, \dots, V_{19} are expressed as follows:

$$V_9 = K_{19} \oplus CL_1 \oplus CH_4 \oplus S(V_5) \oplus S(V_{17} \oplus CL_4 \oplus CH_7 \oplus S(V_7)) \tag{20}$$

$$V_8 = K_7 \oplus CH_1 \oplus S(V_3) \oplus S(K_{19} \oplus CL_1 \oplus CH_4 \oplus S(V_5)) \tag{21}$$

$$V_4 = K_{14} \oplus S(V_1) \oplus S(K_7 \oplus CH_1 \oplus S(V_3)) \tag{22}$$

$$V_2 = K_2 \oplus S(V_{16}) \oplus S(K_{14} \oplus S(V_1)) \tag{23}$$

$$V_{12} = K_9 \oplus S(V_{17}) \oplus S(K_2 \oplus S(V_{16})) \oplus CL_9 \oplus CH_{12} \oplus S(V_{17} \oplus CL_4 \oplus CH_7 \oplus S(V_7) \oplus S(V_{16} \oplus CL_7 \oplus CH_{10} \oplus S(V_{11}))) \tag{24}$$

$$V_{18} = K_{16} \oplus S(K_9 \oplus S(V_{17})) \tag{25}$$

$$V_{10} = K_4 \oplus S(K_{16}) \oplus CL_3 \oplus CH_6 \oplus S(V_8) \oplus S(V_{18} \oplus CL_6 \oplus CH_9 \oplus S(V_9)) \tag{26}$$

$$V_{14} = K_{15} \oplus CH_3 \oplus S(V_4) \oplus S(K_4 \oplus S(K_{16}) \oplus CL_3 \oplus CH_6 \oplus S(V_8)) \tag{27}$$

$$V_6 = K_3 \oplus S(V_2) \oplus S(K_{15} \oplus CH_3 \oplus S(V_4)) \tag{28}$$

$$V_{15} = V_1 \oplus CL_{10} \oplus CH_{13} \oplus S(A \oplus CL_5 \oplus CH_8 \oplus S(V_{14}) \oplus S(V_{13})) \tag{29}$$

$$V_{11} = K_0 \oplus CL_2 \oplus CH_5 \oplus S(V_6) \oplus S(A \oplus CL_5 \oplus CH_8 \oplus S(V_{14})) \tag{30}$$

$$V_7 = B \oplus S(K_0 \oplus CL_2 \oplus CH_5 \oplus S(V_6)) \tag{31}$$

$$V_5 = K_{18} \oplus S(V_{19}) \oplus S(B) \tag{32}$$

$$V_{13} = C \oplus CL_8 \oplus CH_{11} \oplus S(V_{10}) \quad (33)$$

$$V_3 = K_6 \oplus S(C) \oplus S(K_{18} \oplus S(V_{19})) \quad (34)$$

$$V_1 = K_{13} \oplus S(A) \oplus S(K_6 \oplus S(C)) \quad (35)$$

$$V_{16} = K_1 \oplus S(K_0) \oplus S(K_{13} \oplus S(A)) \quad (36)$$

$$V_{17} = K_8 \oplus S(K_1 \oplus S(K_0)) \quad (37)$$

$$V_{19} = K_{17} \oplus S(V_{18}) \oplus S(K_{10} \oplus S(K_9 \oplus S(V_{17}) \oplus S(K_2 \oplus S(V_{16})))) \quad (38)$$

$$B = K_{11} \oplus CH_2 \oplus S(K_{10} \oplus S(K_9 \oplus S(V_{17}) \oplus S(K_2 \oplus S(V_{16})))) \oplus S(K_3 \oplus S(V_2)) \quad (39)$$

$$C = K_5 \oplus S(K_4 \oplus S(K_{16})) \oplus S(K_{17} \oplus S(V_{18})) \quad (40)$$

$$A = K_{12} \oplus S(K_5 \oplus S(K_4 \oplus S(K_{16}))) \quad (41)$$

Therefore, we can compute the values of the variables V_1, \dots, V_{19} directly from Eqs. (1)–(19). Hence, we substitute their values into the Eqs. (20)–(41). Thus, it is easy to obtain the values of $K_{19}, K_7, K_{14}, K_2, K_9, K_{16}, K_4, K_{15}, K_3, A, K_0, B, K_{18}, C, K_6, K_{13}, K_1, K_8$ one by one from Eqs. (20)–(37). Next, we guess the value of K_{10} and obtain the values of $K_{17}, K_{11}, K_5, K_{12}$ from Eqs. (38)–(41).

References

1. 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
2. Beierle, C., Leander, G., Moradi, A., Rasoolzadeh, S.: CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. IACR Trans. Symmetric Cryptology **2019**(1), 5–45 (2019). <https://www.tosc.iacr.org/index.php/ToSC/article/view/7396>
3. Chu, Z., et al.: Improved integral attacks without full codebook. IET Inf. Secur. **12**(6), 513–520 (2018)
4. Daemen, J., Knudsen, L., Rijmen, V.: The block cipher Square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0052343>
5. ElSheikh, M., Tolba, M., Youssef, A.M.: Integral Attacks on Round-Reduced BelT-256. In: Cid, C., Jacobson Jr., M.J. (eds.) Selected Areas in Cryptography - SAC 2018. LNCS, vol. 11349, pp. 73–91. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-10970-7_4
6. ElSheikh, M., Youssef, A.M.: Integral Cryptanalysis of Reduced-Round Tweakable TWINE. Cryptology ePrint Archive, Report 2020/1227 (2020). <https://eprint.iacr.org/2020/1227>
7. Ferguson, N., et al.: Improved cryptanalysis of Rijndael. In: Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44706-7_15
8. Gurobi Optimization, L.: Gurobi Optimizer Reference Manual (2020). <http://www.gurobi.com>
9. Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: Deoxys v1.41. Submitted to CAESAR Competition (2016). <https://competitions.cr.yt.to/round3/deoxysv141.pdf>

10. Knudsen, L., Wagner, D.: Integral cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45661-9_9
11. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21702-9_18
12. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. *J. Cryptology* **24**(3), 588–613 (2011)
13. Peyrin, T., Seurin, Y.: Counter-in-tweak: authenticated encryption modes for tweakable block ciphers. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 33–63. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_2
14. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30539-2_2
15. Sakamoto, K., et al.: Tweakable TWINE: building a tweakable block cipher on generalized feistel structure. In: Attrapadung, N., Yagi, T. (eds.) IWSEC 2019. LNCS, vol. 11689, pp. 129–145. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26834-3_8
16. Sasaki, Yu., Wang, L.: Meet-in-the-middle technique for integral attacks against feistel ciphers. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 234–251. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35999-6_16
17. Sun, L., Wang, W., Wang, M.: MILP-aided bit-based division property for primitives with non-bit-permutation linear layers. *Cryptology ePrint Archive, Report 2016/811* (2016). <https://eprint.iacr.org/2016/811>
18. Sun, L., Wang, W., Wang, M.: Automatic search of bit-based division property for ARX ciphers and word-based division property. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 128–157. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_5
19. Suzaki, T., Minematsu, K.: Improving the generalized feistel. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 19–39. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13858-4_2
20. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE: A lightweight, versatile block cipher. In: ECRYPT Workshop on Lightweight Cryptography, pp. 28–29. Belgium (2011)
21. Todo, Y., Morii, M.: Bit-based division property and application to SIMON family. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 357–377. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_18
22. Tolba, M., ElSheikh, M., Youssef, A.M.: Impossible differential cryptanalysis of reduced-round tweakable TWINE. In: Nitaj, A., Youssef, A. (eds.) AFRICACRYPT 2020. LNCS, vol. 12174, pp. 91–113. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51938-4_5
23. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_24
24. Zhang, H., Wu, W.: Structural evaluation for generalized Feistel structures and applications to LBlock and TWINE. In: Biryukov, A., Goyal, V. (eds.) INDOCRYPT 2015. LNCS, vol. 9462, pp. 218–237. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26617-6_12



RiCaSi: Rigorous Cache Side Channel Mitigation via Selective Circuit Compilation

Heiko Mantel, Lukas Scheidel, Thomas Schneider, Alexandra Weber^(✉),
Christian Weinert, and Tim Weißmantel

Technical University of Darmstadt, Darmstadt, Germany
{mantel, weber, weissmantel}@mais.informatik.tu-darmstadt.de,
{scheidel, schneider, weinert}@encrypto.cs.tu-darmstadt.de

Abstract. Cache side channels constitute a persistent threat to crypto implementations. In particular, block ciphers are prone to attacks when implemented with a simple lookup-table approach. Implementing crypto as software evaluations of circuits avoids this threat but is very costly.

We propose an approach that combines program analysis and circuit compilation to support the selective hardening of regular C implementations against cache side channels. We implement this approach in our toolchain RiCaSi. RiCaSi avoids unnecessary complexity and overhead if it can derive sufficiently strong security guarantees for the original implementation. If necessary, RiCaSi produces a circuit-based, hardened implementation. For this, it leverages established circuit-compilation technology from the area of secure computation. A final program analysis step ensures that the hardening is, indeed, effective.

1 Introduction

Cache side channels are unintended communication channels of programs. Cache-side-channel leakage might occur if a program accesses memory addresses that depend on secret information like cryptographic keys. When these secret-dependent memory addresses are loaded into a shared cache, an attacker might deduce the secret information based on observing the cache.

Such cache side channels are particularly dangerous for implementations of block ciphers, as shown, e.g., by attacks on implementations of DES [58, 67], AES [2, 11, 57], and Camellia [59, 67, 73]. A key reason why block-cipher implementations are vulnerable to cache-side-channel attacks is that they traditionally use secret-dependent accesses to lookup tables in memory. For instance, the original AES specification [20] recommends lookup tables to increase performance. Such lookup-table-based AES implementations are still available in many crypto libraries, including, e.g., OpenSSL [55] and mbedTLS [7].

To avoid cache-side-channel leakage, block ciphers can be implemented as circuits that are evaluated in software, e.g., using the bitslicing technique [12, 35, 47]. For instance, Matsui and Nakajima [47], as well as Käsper and

Schwabe [35] argue why their circuit-based AES implementations are side-channel resistant.

Manually developing circuit-based implementations from algorithm specifications is costly and error-prone due to the huge gap between the two levels of abstraction. Moreover, to run software-based circuits in a real-world setting, additional code is needed, e.g., to initialize the inputs. Since this additional code is a potential source of leakage, its development requires a high level of rigor.

Unfortunately, there is currently no end-to-end tool support for this complex task: Existing tools for generating circuit-based crypto implementations require the input specification to be already at the level of a circuit description [9, 48]. Conversely, existing tools for high-level synthesis that operate, e.g., on ANSI C or SystemC programs do not generate software. Instead, they transpile code to hardware description languages like Verilog or VHDL [51], from which logic synthesis tools (e.g., [62]) can derive FPGA configurations or ASIC designs.

We address this open problem by proposing an approach that hardens high-level C implementations by translating them into circuit-based software implementations. Our approach applies the hardening selectively, based on automatic quantitative program analysis. To support the translation to circuit format, our approach leverages existing compiler infrastructures from the area of secure computation, where circuit compilers, e.g., [15, 32, 40, 43, 61, 72], are used to generate circuit descriptions that obviously evaluate functions on private inputs via homomorphic encryption [27] or interactive cryptographic protocols [28, 71].

We implement our approach in our toolchain RiCaSi, which takes as input a regular C implementation (e.g., of a block cipher) and outputs a circuit-based x86 binary together with a reliable quantitative security guarantee with respect to cache-side-channel leakage. Naturally, these security guarantees are based on established formal models. RiCaSi builds on the circuit compiler HyCC [15] and the program analysis tool CacheAudit [25], augmented with novel implementations and extensions required for the toolchain integration. Supplementary downloads are freely available at www.mais.informatik.tu-darmstadt.de/ricasi.html.

We evaluate RiCaSi across lookup-table-based AES implementations from the libraries OpenSSL [55], mbedTLS [7], Nettle [49] and LibTomCrypt [41], and across implementations of DES [52], 3DES, and Camellia [4] from mbedTLS. RiCaSi is easily applicable to all of these implementations. Moreover, it successfully improves their level of cache-side-channel security. For instance, the analysis integrated in RiCaSi derives an upper bound of 73.82 bit on the amount of information that the original OpenSSL AES might leak to an access-based cache side-channel attacker. After the conversion to a circuit-based implementation, this leakage bound drops to 0 bit. These upper bounds are based on rigorous program analysis and, hence, constitute reliable security guarantees.

Overall, we summarize our contributions as follows:

1. We present RiCaSi, a toolchain that semi-automatically produces circuit-based implementations of block ciphers with corresponding quantitative security guarantees on cache side-channel leakage.

2. We evaluate RiCaSi across implementations of AES, DES, 3DES, and Camellia, demonstrating the effectiveness of our approach by obtaining 0 bit upper leakage bounds for previously vulnerable implementations.
3. We furthermore evaluate the run-time and storage overhead induced by RiCaSi, demonstrating its practicality for security-critical applications.

2 Preliminaries

2.1 The Block Ciphers AES, DES and Camellia

AES. The Advanced Encryption Standard (AES) [53] is a block cipher that encrypts 128 bit message blocks using a symmetric secret key of size 128, 192, or 256 bit. To this end, AES creates so-called round keys from the secret key. The first round key is added to the message block using bitwise XOR. The remaining round keys are used to transform the block in multiple rounds.

The original AES proposal [20] suggests optimizing performance by precomputing the results of the transformation rounds for all possible inputs and storing them in lookup tables. Then, one simply needs to look up the transformation result from the table at the index corresponding to the current round input. The round inputs are the round key and the current state of the transformed message. Implementations that follow this table-based technique are prone to cache side-channel attacks: The indices of the table accesses and, hence, the addresses of the accessed memory locations depend on the secret message and round keys. If a memory entry is loaded into a cache that is shared with an attacker, the attacker might notice the presence of the entry in the cache and deduce secret information. He might even recover the entire secret key [2, 5, 6, 11, 31, 36].

DES. The Data Encryption Standard (DES) [52] is a block cipher that encrypts 64 bit message blocks using a symmetric secret key of size 56 bit. Triple DES (3DES) is an extension for a key size of 168 bit, essentially performing three DES encryptions sequentially using three 64 bit substrings of the 3DES key as DES keys. Both DES and 3DES are deprecated [54], but they are still part of many common crypto libraries like mbedTLS [7] and OpenSSL [55].

Implementation of DES and 3DES might be susceptible to cache side-channel attacks. DES keys can, e.g., be recovered based on the cache misses encountered by implementations that use eight lookup tables (S-Boxes) for substitutions [67]. Such an implementation with eight lookup tables is, e.g., available in mbedTLS.

Camellia. Camellia [4] is a block cipher that encrypts 128 bit message blocks with symmetric secret 128, 192, or 256 bit keys in transformation rounds. Like AES and DES, Camellia uses round keys in each transformation round.

There are multiple techniques for cache attacks on implementations of Camellia that use lookup tables (S-Boxes). The Camellia secret key can, e.g., be recovered from an implementation with four tables using cache-access patterns obtained from power measurements [59]. Access-driven cache attacks can also be used to recover keys from a table-based Camellia implementation [73]. Table-based implementations are available, e.g., in OpenSSL [55] and mbedTLS [7].

2.2 Boolean Circuits for Secure Computation

Secure computation techniques make it possible to involve untrusted parties in the processing of private data. More specifically, homomorphic encryption allows one to outsource computation on private data to untrusted third parties [27]. In contrast, in secure two- or multi-party computation, two or more mutually distrusting parties jointly and interactively compute on private data [28,71].

Secure computation techniques obviously compute publicly known functions expressed as combinatorial Boolean and/or arithmetic circuits. Boolean circuits are composed of AND and XOR gates, whereas arithmetic circuits consist of addition and multiplication gates. Both types of circuits are functionally complete when having access to constants, i.e., they can represent arbitrary computable functions. A Turing machine T with input length n can be expressed as a circuit of size $\tilde{O}(t(T, n))$, where $t(T, n)$ denotes the running time of T on input length n [30].

As noted in [26], the evaluation of Boolean and arithmetic circuits as done in secure computation is inherently secure against a wide range of software side-channel attacks. This is due to the fact that every possible branch of the function represented by such a circuit is executed in parallel and that the memory accesses performed by such circuit implementations do not depend on input data.

Unfortunately, designing circuits from high-level function descriptions is complex and requires tool support. Moreover, for secure computation, expert knowledge about the underlying protocols is required to achieve efficient results.

In hardware design, there exist academic as well as commercial high-level synthesis tools that automatically transpile, e.g., ANSI C or SystemC code to hardware description languages like Verilog or VHDL [51]. Via established logic synthesis tools (e.g., [62]) that output FPGA configurations or ASIC designs, it is furthermore possible to go to hardware level. Logic synthesis tools have also been adapted for secure computation by providing customized ASIC cell libraries and optimization parameters as well as algorithms [21,61,63,64].

Being much more convenient for regular software developers, a line of research has focused on creating optimized compilers that directly transform ANSI C programs to basic (Boolean) circuit representations that can easily be evaluated in software, e.g., by secure computation frameworks like ABY [22]. State-of-the-art in this domain is HyCC [15], the successor of the CBMC-GC compiler [32], which in turn is based on the bounded model checker CBMC [17].

HyCC provides optimizations like automated parallelization of concurrent code, logic minimization, loop unrolling, and minimization of the resulting circuits. However, in this work we target only size-optimized Boolean circuits and hence do not use the computationally expensive optimization steps of HyCC.

2.3 Program-Analysis Approach

To quantify the leakage of x86 binaries through cache side channels, we use a combination of information theory and abstract interpretation. This approach was first established in [38], later extended and then implemented in

the tool CacheAudit [25], of which multiple variants have been developed (e.g., [13, 24, 46]). We build on CacheAudit and extend it with support for additional language features where necessary. Below, we describe the underlying approach in more detail.

We model a cache side channel as a deterministic, discrete, memoryless channel from an input alphabet (random variable X) to an output alphabet (random variable Obs). The min-entropy $H_\infty(X) = -\log_2 \max_i p(x_i)$ of X captures the uncertainty an attacker has about the secret input if the probability for each input x_i is $p(x_i)$ [60]. The conditional min-entropy $H_\infty(X|Obs) = -\log_2 \sum_{j=1}^{|Obs|} p(obs_j) \cdot \max_i \frac{p(obs_j|x_i) \cdot p(x_i)}{p(obs_j)}$ captures the attacker’s remaining uncertainty after observing the channel output, where output obs_j occurs for secret x_i with probability $p(obs_j|x_i)$ and occurs overall with probability $p(obs_j)$. The information that an output reveals about the input is modeled by the min-entropy leakage $H_\infty(X) - H_\infty(X|Obs)$, which is upper bounded by $\log_2 |Obs|$ bit [39, 60].

Let X be the set of possible secret inputs (secret key and message) and Obs be the possible observations of a cache-side-channel attacker. We compute cache side-channel leakage bounds as $\log_2 |Obs^{\overline{\mathcal{D}}}|$, where $Obs^{\overline{\mathcal{D}}}$ is an overapproximation of the reachable observations Obs . The overapproximation makes the analysis feasible and is done using abstract interpretation [19]. More concretely, we overapproximate the actual possible execution states \mathcal{D} by more abstract execution states $\overline{\mathcal{D}}$ and the actual semantics $\text{upd}_{\mathcal{D}} : \mathcal{D} \times \mathcal{I} \rightarrow \mathcal{D}$ of instruction set \mathcal{I} by an abstract semantics $\text{upd}_{\overline{\mathcal{D}}} : \overline{\mathcal{D}} \times \mathcal{I} \rightarrow \overline{\mathcal{D}}$. We then compute the reachable abstract observations according to $\text{upd}_{\overline{\mathcal{D}}}$ and count the number of actual observations they represent. We take the logarithm to obtain the leakage bound $\log_2 |Obs^{\overline{\mathcal{D}}}|$.

We consider four models of cache side-channel attackers, i.e., four variants of Obs : (1) Attackers under the model *acc* can deduce which memory entries are cached in a shared cache after the victim program is executed, (2) attackers under *accd* can deduce the number of memory entries the victim loaded into each cache set of such a shared cache, (3) attackers under *trace* can deduce the trace of cache hits and cache misses that occurred during the victim-program execution, and (4) attackers under *time* can deduce the execution time of a victim-program execution (modeled by fixed durations for cache hits, misses, and other steps).

3 The RiCaSi Toolchain

The goal of RiCaSi is to allow developers to obtain x86 binaries from regular and potentially vulnerable C code that come with quantitative security guarantees with respect to cache side channels. The high-level overview of our toolchain and its workflow is depicted in Fig. 1.

First, the C code provided by the user (e.g., a block-cipher implementation) is compiled to an x86 binary (e.g., with GCC) and analyzed with our extended version of CacheAudit (cf. Sect. 3.5). If the resulting upper bound on the cache-side-channel leakage of the binary is below an acceptable threshold, the binary can be used securely and RiCaSi terminates.

In case the threshold leakage is exceeded, RiCaSi compiles the C code into a circuit representation, which in turn is compiled into an x86 binary for further analysis. For this, we first preprocess the C code to substitute constructions currently not supported by existing circuit compilers (cf. Sect. 3.2), e.g., with respect to memory management and the passing of parameters.

The resulting C code is then compiled with the state-of-the-art circuit compiler HyCC [15] (cf. Sect. 3.3). For transforming the resulting circuit representation back to C code, we implement our own tool in Python (cf. Sect. 3.4).

After compilation to an x86 binary, we perform a second round of analysis with our extension of CacheAudit (cf. Sect. 3.5). Here, the expected output is an improved security guarantee in the form of an upper leakage bound that lies below the acceptable threshold or is even equal to 0 bit leakage.

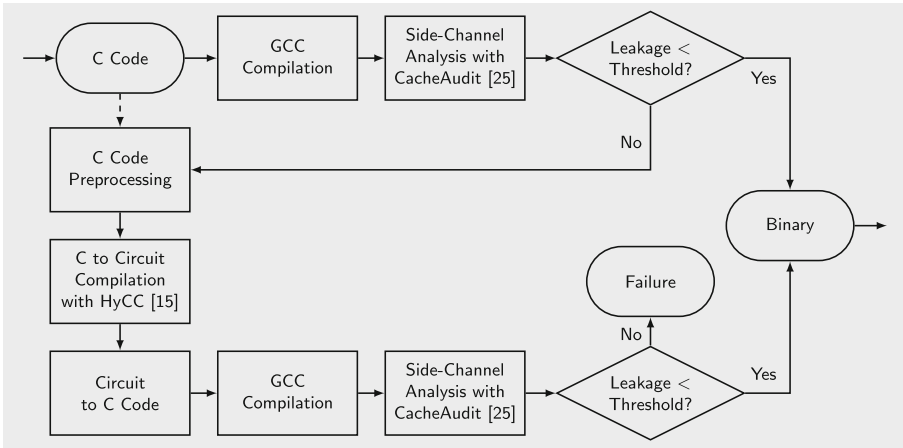


Fig. 1. Overview of our RiCaSi toolchain and workflow.

In the following, we detail the individual steps of the RiCaSi toolchain at the running example of a lookup-table-based implementation of AES encryption from OpenSSL (cf. Listing 1). The original implementation is vulnerable to cache side-channel attacks because it accesses lookup tables (e.g., table `Te0` as shown in Listing 1) at round-key dependent indices.

```

static const u32 Te0[256] = {0xc66363a5U, ...}; // lookup table
...
void AES_encrypt(const unsigned char *in, ...) {...
    s0 = GETU32(in) ^ rk[0]; ... // initial round
    t0 = Te0[s0 >> 24] ^ ...; // secret-dependent memory access
...}
  
```

Listing 1. Excerpt of OpenSSL AES encryption.

3.1 Initial Side-Channel Analysis

In the first step of RiCaSi, we apply automatic program analysis to determine whether the input implementation can be used as is or whether any hardening against cache side channels is required. To this end, we derive quantitative security guarantees for the x86 binary corresponding to the given implementation. More concretely, we compute upper bounds on the cache side-channel leakage of the binary and compare them to the threshold for the desired level of security. If non-zero bounds are not acceptable, the threshold can be set to zero. Technically, we use a combination of information theory and abstract interpretation, implemented in the tool CacheAudit. The tool takes an x86 binary and outputs bounds on the min-entropy leakage to the attacker models *acc*, *accd*, *trace* and *time* (cf. Sect. 2.3).

If the resulting leakage bounds lie below the desired leakage threshold, no hardening is required and unnecessary overhead can be avoided. If the leakage bounds are too high, we proceed with the preprocessing for circuit compilation.

Listing 2 shows the leakage bounds for OpenSSL AES from Listing 1. The bounds guarantee, for instance, that at most 73.83 bit are leaked to an attacker under the model *acc* (cf. Line 2 in Listing 2) and at most 70.34 bit are leaked to an attacker under *accd* (cf. Line 3). These bounds are rather weak security guarantees and would likely exceed the acceptable threshold leakage for most applications such that further steps of RiCaSi would be applied.

```

...
Number of valid cache config. (shared memory): ... (73.820808 bits)
Number of valid cache config. (disj. memory): ... (70.339850 bits)
# traces: ..., 280.000000 bits
# times: ..., 8.134426 bits
Analysis took 18 seconds.

```

Listing 2. Excerpt of analysis results for OpenSSL AES.

3.2 C Code Preprocessing

The “C-to-circuit” compilation as provided by HyCC [15] comes with several limitations regarding the processable source code. To avoid compilation issues, we manually apply several preprocessing steps to make existing implementations compatible. Although these steps are targeted towards our case studies (cf. Sect. 4), they might be of independent interest and worth to fully automate, as they can be applied to make the regular usage of HyCC more convenient.

1. Especially when compiling code that depends on an extensive library, it is best to first bundle all required methods in a single file. If possible, all method calls are replaced by inlining the required code into a method named `mpc_main`. Otherwise, debugging compilation errors becomes infeasible.
2. Global and static variables are not supported by HyCC. They must instead be declared in the main method (cf. the declaration of `Te0` in Listing 3).

3. Memory management via `malloc` and `calloc` is not supported by HyCC. Often, it is sufficient to declare arrays with fixed size instead. In many cases, it is also possible to simply remove such statements as the compiler can determine array sizes from later assignments. However, dealing with such memory management issues was not required for any of our case studies. HyCC also does not support passing arrays or pointers in method headers. This can be circumvented by splitting arrays into single variables, which are passed instead (cf. passing the plaintext data `in01` in Listing 3).

The preprocessed OpenSSL AES encryption code is shown in Listing 3.

```
int mpc_main(unsigned char in01, ...) {... // inputs split in bytes
const unsigned char in[16] = {in01, ...}; // reconstruct inputs
const u32 Te0[256] = {0xc66363a5U, ...}; // table declaration
...}
```

Listing 3. Excerpt of preprocessed OpenSSL AES encryption.

3.3 C Code to Circuit Compilation

The compilation of the preprocessed C code to a circuit description happens through a straightforward application of HyCC [15]. For the compilation, the C code is first transformed into a “goto code” intermediate representation, loops are unrolled, variables are split into single bits, and operations over those bits are expressed as Boolean functions [32]. As briefly described in Sect. 2.2, HyCC also performs several optimizations like circuit minimization.

However, in comparison to the regular usage of HyCC, several of the most computationally expensive steps can be skipped. This is because here we target only the creation of size-optimized Boolean circuits and do not consider depth-optimized Boolean or arithmetic circuits (which are beneficial for some interactive secure computation protocols [28]).

Therefore, for our purpose, HyCC does not need to decompose the code into separate modules and compile each module into multiple different types of circuits. We can also skip the final step where HyCC tries to heuristically optimize the total cost for secure computation protocols by finding the best possible combination of different types of circuits and protocols.

3.4 Circuit to Binary Compilation

In the following, we describe how the HyCC circuit output is translated into C code and further compiled into an x86 binary.

The circuit output produced by HyCC is by default in a binary format to be processed by the ABY MPC framework [22] via a specialized adapter. To facilitate further processing without ABY, we use a HyCC export functionality for conversion into the human-readable and widely used BRISTOL circuit format [66]. In the BRISTOL format, every line of the circuit description

file declares the type of one gate as well as the number and the identifiers of the gate's input and output wires. The supported gate types are AND, XOR, and INV (inversion). The header of the circuit description specifies the total number of gates, the total number of wires, and circuit input as well as output wires. The HyCC output in the BRISTOL format for OpenSSL AES is shown in Listing 4.

```
490425 490809
384 0 160
...
2 1 121 377 385 XOR // XOR gate in BRISTOL representation
1 1 385 386 INV // INV gate in BRISTOL representation
2 1 384 386 387 AND // AND gate in BRISTOL representation
...
```

Listing 4. HyCC circuit for OpenSSL AES encryption in BRISTOL format.

We implemented a converter tool in Python to translate BRISTOL circuit description files into C source code. The converter is controlled via a configuration file that, besides the circuit name and file, specifies input and output types, and which external libraries (e.g., `stdio.h`) should be included.

The converter first declares a variable for each wire and disassembles the specified inputs into the respective circuit input wires. It then iterates through each line of the circuit description and inserts the respective C instruction for performing the specified gate operation (e.g., `&` for AND gates) on the variables corresponding to the gate input and output wires. This is possible because the gates in the BRISTOL circuit format are ordered topologically, i.e., all input wires for each gate have been assigned before. Finally, the circuit output is assembled in the configured type from the circuit output wires. The converter output for our OpenSSL AES encryption example is shown in Listing 5.

```
int openssl_aes_enc(int in01, ...) {
    unsigned char w0, ..., w490808;
    int inbits01[8] = split(in01);
    w0 = inbits01[0];
    ...
    w385 = w121 ^ w377; // C code for gate 2 1 121 377 385 XOR
    w386 = !w385; // C code for gate 1 1 385 386 INV
    w387 = w384 & w386; // C code for gate 2 1 384 386 387 AND
    ...
}
```

Listing 5. Excerpt of OpenSSL AES encryption in circuit-style C.

The resulting C code file can then be compiled into an x86 binary using, for example, the GCC compiler. It is also possible to integrate the produced C code with another application before compilation, or to modify the code, e.g., to include further input and output processing.

3.5 Final Side-Channel Analysis

To ensure that no potential for cache side-channel leakage remains in the final circuit binary, we perform an additional program analysis step. To this end, we apply a variant of the tool CacheAudit that we extended for the purpose of analyzing circuit binaries. Our variant of CacheAudit augments the prior version in two directions: support for large control-flow graphs and support for additional x86 opcodes.

Circuit-based binaries are significantly larger than regular binaries because all individual gates are encoded in the assembly code. Since CacheAudit was not intended for the analysis of binaries with large basic blocks, its parser quickly runs into overflows when trying to build a control-flow graph for the studied circuit-based binaries. By rewriting the corresponding parts of the CacheAudit implementation in a tail-recursive style, we now avoid this issue.

Furthermore, circuit-based binaries use x86 opcodes that did not occur in the binaries that have been analyzed with CacheAudit before. In particular, the comparison instructions with opcodes `0xA8` and `0xF7/0` occur in the binaries. We added support for both instructions to CacheAudit.

Our resulting variant of CacheAudit can be successfully applied to all circuit-based binaries in our evaluation and is of independent interest.

Listing 6 shows an excerpt of the analysis results for the x86 binary corresponding to the circuit-compiled variant of OpenSSL AES from Listing 5. In this example, the resulting leakage bounds are 0 bit across all four attacker models (cf. Line 2 in Listing 6 for *acc*, Line 3 for *acd*, Line 4 for *trace* and Line 5 for *time*). That is, the circuit-compiled binary does not leak secret information through cache side channels to attackers under any of these attacker models.

```

...
Number of valid cache config. (shared memory): 1, (0.000000 bits)
Number of valid cache config. (disj. memory): 1, (0.000000 bits)
# traces: 1, 0.000000 bits
# times: 1.000000, 0.000000 bits
Analysis took 185392 seconds.

```

Listing 6. Excerpt of analysis results for circuit-compiled OpenSSL AES.

Note that circuit compilation does not inevitably lead to 0 bit leakage bounds. Since side channels are vulnerabilities at the level of implementation details, it is crucial to ensure that the hardening is effective in all details. In an intermediate version of RiCaSi we had accidentally introduced potential side-channel leakage in the circuit-to-C compilation step: our initialization of the circuit inputs was not constant-time. With the final program-analysis step of RiCaSi, we detected the mistake due to unexpectedly high leakage bounds for the generated binary. We then adapted our circuit-to-C compilation tool accordingly. As shown in Listing 6, the hardening with RiCaSi is now effective in all details, leading to 0 bit leakage bounds for the resulting binary.

4 Evaluation of Cache-Side-Channel Security

We evaluate the applicability of RiCaSi and the benefit it provides in terms of cache side-channel security guarantees in two dimensions.

We first consider a range of lookup-table-based AES implementations: an implementation from OpenSSL [55] that uses four lookup tables of size 1 kB, an implementation from mbedTLS [7] (a library used, e.g., by cURL [65] and OpenVPN [56]) that uses four 1 kB tables and a 0.25 kB S-Box, an alternative implementation with lookup tables and an S-Box from Nettle [49], and one implementation from the library LibTomCrypt [41] that uses eight 1 kB lookup tables.

In the second step, we broaden the evaluation to implementations of other block ciphers. We consider implementations of three additional block ciphers from the library mbedTLS: Camellia, DES, and 3DES.

4.1 RiCaSi for AES Implementations

We analyze the sequence of the key-generation and encryption functions from the respective AES implementations, applied to a 256 bit key and 128 bit plaintext. We configure mbedTLS without x86 VIA PadLock instructions because we are interested only in the software AES implementation. We configure LibTomCrypt to omit assert statements with indirect jumps to make the computation of security guarantees with state-of-the-art program analysis feasible. The details on the configurations that we used are summarized in Table 1.

Table 1. AES implementations inspected in our case study.

Library	Version	Configuration	Analyzed functions
OpenSSL	1.1.1d	default	<code>AES_set_encrypt_key</code> , <code>AES_encrypt</code>
mbedTLS	2.16.5	removed <code>MBEDTLS_PADLOCK_C</code>	<code>mbedtls_aes_init</code> , <code>mbedtls_aes_setkey_enc</code> , <code>mbedtls_aes_encrypt</code> , <code>mbedtls_aes_free</code>
Nettle	3.5	default	<code>aes256_set_encrypt_key</code> , <code>aes256_encrypt</code>
LibTomCrypt	1.18.2	<code>ARGTYPE</code>	<code>rijndael_enc_setup</code> , <code>rijndael_enc_ecb_encrypt</code>

To compute guarantees for the cache side-channel security of these implementations before and after circuit compilation, we use our extension of the program analysis tool CacheAudit as described in Sect. 3.5.

Our analysis with CacheAudit and the resulting security guarantees focus on one single level of cache. This is a common simplification of modern multi-level cache hierarchies that is frequently applied in cache side-channel quantification [13, 24, 25, 46]. In our analysis, we consider an 8-way set associative

cache with 64 cache sets and a line size of 64 Bytes with the PLRU cache line replacement strategy. This reflects, e.g., the L1 data caches of the Intel Skylake architecture [33, Table 2–4], [1] and the AMD Zen2 architecture [3].

As a baseline for our evaluation, we compute upper bounds on the cache side-channel leakage of the original, vulnerable AES implementations. We then harden the implementations using RiCaSi. In the final analysis step of RiCaSi, CacheAudit is applied again to compute cache-side-channel leakage bounds for the hardened implementations. In both cases, we consider the 32 bit x86 binaries obtained from the C implementations using gcc version 5.4.0.

Baseline Results. Our baseline analysis results across the AES implementations and side-channel attacker models described in Sect. 3.5 are shown on the left side of Table 2. We round the leakage bounds to two decimal places.

Table 2. AES leakage bounds in [bit] before (left) and after (right) RiCaSi.^a

Cipher	Attacker Model				Cipher	Attacker Model			
	<i>acc</i>	<i>accd</i>	<i>trace</i>	<i>time</i>		<i>acc</i>	<i>accd</i>	<i>trace</i>	<i>time</i>
OpenSSL	73.82	70.34	280.00	8.13	OpenSSL	0.00	0.00	0.00	0.00
mbedtls	88.09	81.55	287.00	8.17	mbedtls	0.00	0.00	0.00	0.00
Nettle	85.93	78.55	299.00	8.23	Nettle	0.00	0.00	0.00	0.00
LibTomCrypt	204.03	143.43	274.00	8.10	LibTomCrypt	0.00	0.00	0.00	0.00

^aFor homogeneity across tables, we use the full display format also for all-zero tables.

CacheAudit yields rather high leakage bounds, between 70.34 bit and 299.00 bit for the attacker models *acc*, *accd* and *trace* across all libraries. For the attacker model *time*, the bounds are lower and lie between 8.10 bit and 8.23 bit. For instance, the *time* leakage bound for OpenSSL AES is 8.13 bit. This means that one execution of this AES binary leaks at most 2.12% of the 384 secret bits (256 bit key and 128 bit plaintext) to an attacker under the model *time*.

The high leakage bounds for *acc*, *accd* and *trace* are rather weak security guarantees. That is, for the attacker models *acc*, *accd* and *trace*, the level of security on which we can rely is rather low. Even for the attacker model *time*, we do not obtain guarantees for the complete absence of leakage.

The high bounds are not surprising because all analyzed binaries belong to lookup-table-based implementations that use secret-dependent memory accesses. Next, we evaluate how effective RiCaSi is in hardening the implementations.

Results for RiCaSi. The leakage bounds for the circuit-based binaries produced by RiCaSi are shown on the right-hand side of Table 2. Note that the upper bounds on the leakage are 0 bit across all implementations and attacker models. That is, no information is leaked to attackers under the four models.

While the 0 bit leakage bounds might not be surprising at first sight, recall that they play a central role in RiCaSi. If any detail of the circuit compilation and translation failed, as in our prior implementation (cf. Sect. 3.5), we would

spot this here. With 0 bit bounds, we can be sure that the hardening with RiCaSi is effective in all implementation details.

Table 3. DES, 3DES and Camellia implementation inspected in our case study.

Cipher	Key length	Plaintext length	Analyzed functions
Camellia	256 bit	128 bit	<code>mbedtls_camellia_init</code> , <code>mbedtls_camellia_setkey_enc</code> , <code>mbedtls_camellia_crypt_ecb</code> , <code>mbedtls_camellia_free</code>
DES	64 bit	64 bit	<code>mbedtls_des_init</code> , <code>mbedtls_des_setkey_enc</code> , <code>mbedtls_des_crypt_ecb</code> , <code>mbedtls_des_free</code>
3DES	128 bit	64 bit	<code>mbedtls_des3_init</code> , <code>mbedtls_des3_set2key_enc</code> , <code>mbedtls_des3_crypt_ecb</code> , <code>mbedtls_des3_free</code>

4.2 RiCaSi for Block Ciphers from mbedtls

For each of the three block ciphers Camellia, DES, and 3DES, we analyze the respective sequence of functions to initialize the data structures, compute the key schedule, perform the encryption and free the data structures from mbedtls version 2.16.5. The details, including key and plaintext lengths, are described in Table 3. We use the same CacheAudit variant and configuration as in Sect. 4.1.

Table 4. mbedtls leakage bounds in [bit] before (left) and after (right) RiCaSi.

Cipher	Attacker Model				Cipher	Attacker Model			
	<i>acc</i>	<i>accd</i>	<i>trace</i>	<i>time</i>		<i>acc</i>	<i>accd</i>	<i>trace</i>	<i>time</i>
AES	88.09	81.55	287.00	8.17	AES	0.00	0.00	0.00	0.00
Camellia	28.50	25.75	242.00	7.92	Camellia	0.00	0.00	0.00	0.00
DES	38.40	37.75	141.00	7.16	DES	0.00	0.00	0.00	0.00
3DES	52.20	48.34	416.00	8.70	3DES	0.00	0.00	0.00	0.00

Baseline Results. The leakage bounds for the original block-cipher implementations from mbedtls (including mbedtls AES for comparison) are shown on the left side of Table 4. The *acc* and *accd* leakage bounds for Camellia, DES, and 3DES are lower than the leakage bounds for AES, but still rather high compared to the respective sizes of the secret key and message (384 bit for Camellia,

128 bit for DES, and 192 bit for 3DES). For the attacker models *trace* and *time*, the leakage bounds for 3DES are even higher than those for AES. This might be due to an accumulation of leakage across the DES executions in 3DES.

Again, the high leakage bounds are not surprising given the known cache-side-channel attacks on such implementations described in Sect. 2. Next, we apply RiCaSi to harden the implementations against such attacks. The resulting leakage bounds are shown on the right-hand side of Table 4.

Results for RiCaSi. For all block-cipher implementations hardened with RiCaSi, we are able to derive guarantees for 0 bit leakage for all four cache side-channel attacker models. That is, RiCaSi effectively hardened the implementations against cache side-channel attackers under these models.

Overall, RiCaSi hence supports the hardening not only of AES implementations but also of a broader range of block-cipher implementations. In all cases that we considered in our evaluation, the effectiveness of the hardening was automatically verifiable using the program analysis of CacheAudit.

5 Evaluation of Overhead

Compiling applications into side-channel resistant executables is a one time cost that is quickly amortized over time. However, RiCaSi generates repeated overhead in two aspects, which we evaluate in detail: First, we study how much the size of the circuit-based binaries increases compared to regular compilation results. Then, we evaluate how much the run-time of the side-channel resistant binaries increases compared to the vulnerable counterparts.

5.1 Binary Sizes

In Table 5, we compare the binary sizes of the regular block-cipher implementations to the output produced by RiCaSi. While storage costs nowadays are almost negligible at the given scale, considering the overhead in terms of binary sizes is especially necessary to estimate the additional costs when widely distributing software over the Internet, or for embedded devices.

The results in Table 5 strongly vary among the considered block ciphers. The binary sizes for DES and 3DES, e.g., stay well below 5 MB and have less than factor $5\times$ blow-up. However, binary sizes for AES increase up to about 24 MB, which corresponds to a blow-up of two orders of magnitude. Therefore, we recommend to use RiCaSi mainly on small, highly security critical code sections.

Note that the compilation setup in our case studies was not tailored to optimize the binary sizes. All binaries include debug information. Moreover, while the original AES binaries were linked dynamically, we used static linking for all other binaries to make them self-contained for the program analysis. By dropping dispensable information from the binaries, the sizes could be reduced if necessary.

Table 5. Comparison of binary sizes.

Cipher	Library	Original (in KB)	RiCaSi (in KB)	Overhead
AES	OpenSSL	37.72	23,624.18	626.30×
	Nettle	29.81	23,573.93	790.81×
	LibTomCrypt	56.70	23,623.09	416.63×
	mbedTLS	57.32	23,581.20	411.40×
Camellia	mbedTLS	890.56	11,923.80	13.39×
DES	mbedTLS	891.80	1,408.01	1.58×
3DES	mbedTLS	891.84	3,497.00	3.92×

5.2 Run-Times

We evaluate the run-times of the executables of various block ciphers generated by RiCaSi and compare the resulting overhead to the regular vulnerable executables in Fig. 2. All binaries are executed on one logical core of an Intel Core i9-7960X CPU clocked at 2.8 GHz (with up to 4.2 GHz turbo boost). The stated run-times are averages over 10 executions.

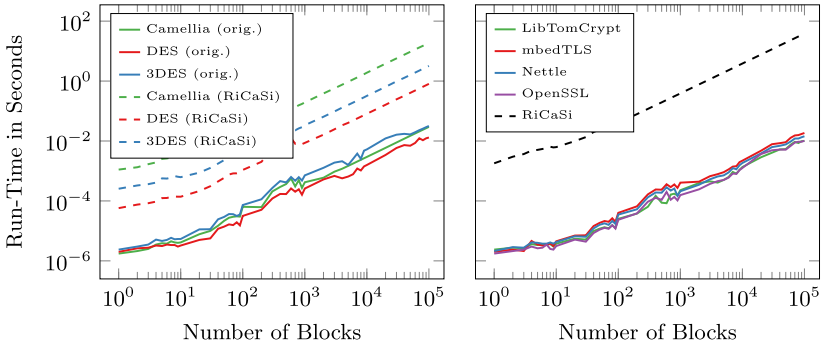


Fig. 2. Comparison of run-times for encrypting an increasing number of blocks with different ciphers. Left: Camellia, DES, 3DES (mbedTLS). Right: different AES implementations; the differences in the respective RiCaSi versions are negligible.

In Fig. 2, we observe about two (DES, 3DES) to three (Camellia, all AES implementations) orders of magnitude overhead when executing the binaries produced by RiCaSi. For encrypting large amounts of data (i.e., in the order of gigabytes) or applications with strict real-time requirements (e.g., Bitlocker), this overhead quickly becomes impractical. However, for processing small or even large amounts of data in high-security contexts without strict real-time requirements, the binaries generated by RiCaSi deliver practical performance.

6 Related Work

6.1 Secure Computation Techniques for Side-Channel Mitigation

So-called one-time programs (OTPs) are studied in [30], which are programs that can be evaluated only on a single input chosen at run-time. The proposed construction is based on a combination of tamper-resistant hardware with Yao’s garbling scheme for Boolean circuits [71]. In this scheme, the gate tables are encrypted and the corresponding keys required for decryption are carried by the circuit wires instead of single bits. Importantly, the nature of the garbled circuit evaluation prevents all potential side-channel leakage.

A variant of this idea was later implemented on FPGAs by [34]. Their performance evaluation observes an overhead of about factor $10^6 \times$ comparing one unprotected AES evaluation in hardware to a provably side-channel resistant hardware-accelerated OTP evaluation. Despite this significant overhead, the authors argue their solution might be reasonable for high-security applications.

In contrast to these works, we provide a generic compiler toolchain for creating and evaluating Boolean (non-garbled) circuits in software. Our performance evaluation shows an overhead of only about factor $10^3 \times$ when comparing regular vulnerable implementations of various block ciphers to circuit-based executables with 0 bit upper leakage bounds with respect to cache side channels.

In [26], Felsen et al. use circuit representations to mitigate side-channel attacks for programs shielded with Intel Software Guard Extensions (SGX) [18]. Intel SGX is a trusted execution environment available in many Intel CPUs that allows one to run so-called *enclaves* in isolation from all other software. However, Intel considers software side channels out of scope for the attacker model, which resulted in many attacks showing the vulnerability especially of enclaves running cryptographic code (e.g., [69]). As a solution, Felsen et al. created an enclave that evaluates Boolean circuits on private inputs provided to the enclave via secure channels [26]. They claim resistance against timing and page-tables as well as cache-based software side channels, but do not provide any analyses for confirmation. Also, they do not provide an integrated solution for obtaining the circuit representations required for their circuit evaluator.

In contrast to [26], we provide an automated way to generate side-channel resistant executables with our toolchain RiCaSi. Most importantly, our approach is backed by formal analyses showing upper bounds of 0 bit on the cache side-channel leakage for various implementations of block ciphers. In the future, RiCaSi could be extended to produce side-channel resistant Intel SGX enclaves.

The concept of Oblivious RAM (ORAM) [29] was introduced to prevent that code can be reverse-engineered from observations about the memory accesses performed by the code. The key idea is to replace each memory access with a sequence of memory accesses that conceals the address of the original memory access. That is, ORAM prevents information leakage via memory accesses without removing these accesses completely. RiCaSi follows the alternative approach of eliminating the memory accesses through circuit compilation.

6.2 Systematic Detection and Assessment of Side-Channel Leakage

Systematic approaches to side-channel security range from qualitative approaches, like type-based techniques [8, 23, 37, 50], to quantitative approaches, like abstraction-based techniques [25, 38, 42] or experiment-based techniques [16, 44, 45]. In the following, we provide an overview of existing qualitative and quantitative approaches with a focus on cache side channels.

Qualitative approaches to cache side-channel detection include, e.g., DATA [70] and CacheD [68]. Both tools check for cache side channels in execution traces. They are intended for debugging and do not provide security guarantees. The tool CaSym [14] soundly verifies LLVM code against cache side channels. While DATA uses statistical methods, CacheD and CaSym use symbolic execution.

Quantitative approaches to cache side-channel assessment include multiple variants of the tool CacheAudit [25]. CacheAudit computes upper bounds on the cache side-channel leakage of x86 binaries using a combination of information theory and abstract interpretation. It has been successfully extended and applied for the analysis of multiple cryptographic implementations, including AES implementations [46], modular exponentiation [24], and lattice-based cryptography [13]. Our work is based on CacheAudit and extends the tool for our purposes with better scalability and additional x86 language coverage.

6.3 Analysis of Side-Channel Leakage in Circuit Implementations

To the best of our knowledge, the closest to our work in combining circuit compilation with side-channel security guarantees are the Usuba compiler [48] and its extension to Tornado [9]. Both, Usuba and Tornado take as input a circuit specification in the Usuba specification language.

Usuba compiles the specification to C code and introduces optimizations like bitslicing. Bitslicing [12] optimizes the performance of circuit-based software implementations by parallelization. To this end, the variables that model the circuit wires are used to store multiple bits instead of just one bit. Applying bitwise operations to these variables will then model the application of the corresponding gate to all bits in parallel.

Tornado augments Usuba and returns an optimized circuit binary that satisfies security guarantees with respect to the register-probing adversary model. That is, the resulting circuit is secure against side-channel adversaries that can probe intermediate values of registers during the execution of the software circuit. To this end, Tornado extends Usuba with support for the masking countermeasure. Masking mitigates side-channel leakage by splitting the secret value into shares that are only meaningful in combination. Moreover, Tornado combines the extended Usuba with the tool TightProve [10] to show that the resulting masked implementation is secure with respect to the register-probing model.

That is, both Usuba and Tornado work at the level of circuits. Both optimize the circuits and Tornado also provides security guarantees. Neither of the tools aims at supporting the development of circuits from high-level specifications.

Overall, Tornado and Usuba are complementary to RiCaSi. Tornado and Usuba focus on optimizing circuits, e.g., by bitslicing. RiCaSi currently uses only one bit of each variable, i.e., does not apply bitslicing. Tornado and Usuba do not support the generation of a circuit specification from a high-level implementation. RiCaSi closes this gap and converts high-level C implementations into circuit-based implementations that are reliably secure against cache side-channel attacks.

7 Conclusion

In this paper, we presented the toolchain RiCaSi, an integrated solution for hardening regular C implementations against cache side channels by transforming them into circuit-based x86 binaries.

RiCaSi applies program analysis to quantify the threat of cache side-channel leakage in a given implementation. Based on the analysis results, the implementation can be hardened selectively and unnecessary costs are avoided. With RiCaSi, we successfully transformed multiple vulnerable crypto implementations (AES from OpenSSL, mbedTLS, Nettle, and LibTomCrypt; Camellia, DES, and 3DES from mbedTLS) into circuit-based binaries with zero-leakage guarantees against four cache attacker models. For these binaries, we observed overhead of up to three orders of magnitude, which is acceptable for critical applications without hard real-time requirements. Overall, RiCaSi performs a selective, effective and affordable hardening of regular C implementations against cache side channels.

In the future, integrating steps for circuit optimization into the toolchain, e.g., the use of vectorized instructions or automated bitslicing as in [48], will be a promising direction to greatly reduce overhead while maintaining the applicability to high-level C implementations and the reliable security guarantees.

Acknowledgments. We thank the anonymous reviewers for their helpful comments. This project was co-funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE. It has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI).

References

1. Abel, A., Reineke, J.: nanoBench: a low-overhead tool for running microbenchmarks on x86 systems. CoRR abs/1911.03282 (2019)
2. Aci mez, O., Koç, Ç.K.: Trace-driven cache attacks on AES (short paper). In: ICICS (2006)
3. Advanced Micro Devices: software optimization guide for AMD family 17h models 30h and greater processors. Publication number: 56305, Revision: 3.02 (2020)
4. Aoki, K., et al.: Specification of Camellia - a 128-bit block cipher, version 2.0 (2001)

5. Apecechea, G.I., Eisenbarth, T., Sunar, B.: S\$A: a shared cache attack that works across cores and defies VM sandboxing - and its application to AES. In: S&P (2015)
6. Apecechea, G.I., Inci, M.S., Eisenbarth, T., Sunar, B.: Wait a minute! A fast, cross-vm attack on AES. In: RAID (2014)
7. ARM Limited: mbedTLS (Version 2.16.5) (2020). <https://tls.mbed.org/download/start/mbedtls-2.16.5-apache.tgz>
8. Barthe, G., Rezk, T., Warnier, M.: Preventing timing leaks through transactional branching instructions. In: QAPL (2006)
9. Belaïd, S., Dagand, P., Mercadier, D., Rivain, M., Wintersdorff, R.: Tornado: automatic generation of probing-secure masked bitsliced implementations. In: EURO-CRYPT (2020)
10. Belaïd, S., Goudarzi, D., Rivain, M.: Tight private circuits: achieving probing security with the least refreshing. In: ASIACRYPT (2018)
11. Bernstein, D.J.: Cache-timing attacks on AES. University of Illinois at Chicago, Technical report (2005)
12. Biham, E.: A fast new DES implementation in software. In: FSE (1997)
13. Bindel, N., Buchmann, J.A., Krämer, J., Mantel, H., Schickel, J., Weber, A.: Bounding the cache-side-channel leakage of lattice-based signature schemes using program semantics. In: FPS (2017)
14. Brotzman, R., Liu, S., Zhang, D., Tan, G., Kandemir, M.T.: Casym: cache aware symbolic execution for side channel detection and mitigation. In: S&P (2019)
15. Büscher, N., Demmler, D., Katzenbeisser, S., Kretzmer, D., Schneider, T.: HyCC: compilation of hybrid protocols for practical secure computation. In: CCS (2018)
16. Chothia, T., Kawamoto, Y., Novakovic, C.: A tool for estimating information leakage. In: CAV (2013)
17. Clarke, E.M., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: TACAS (2004)
18. Costan, V., Devedas, S.: Intel SGX explained. ePrint 2016/86 (2016)
19. Cousot, P., Devsot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL (1977)
20. Daemen, J., Rijmen, V.: AES submission document on Rijndael. Version 2 (1999)
21. Demmler, D., Dessouky, G., Koushanfar, F., Sadeghi, A., Schneider, T., Zeitouni, S.: Automated synthesis of optimized circuits for secure computation. In: CCS (2015)
22. Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
23. Dewald, F., Mantel, H., Weber, A.: AVR processors as a platform for language-based security. In: ESORICS (2017)
24. Doychev, G., Köpf, B.: Rigorous analysis of software countermeasures against cache attacks. In: PLDI (2017)
25. Doychev, G., Köpf, B., Mauborgne, L., Reineke, J.: Cacheaudit: a tool for the static analysis of cache side channels. *ACM Trans. Inf. Syst. Secur.* **18**(1) (2015)
26. Felsen, S., Kiss, Á., Schneider, T., Weinert, C.: Secure and private function evaluation with Intel SGX. In: CCSW (2019)
27. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC (2009)
28. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC (1987)
29. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. *J. ACM* **43**(3), 431–473 (1996)

30. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In: CRYPTO (2008)
31. Gullasch, D., Bangerter, E., Krenn, S.: Cache games - bringing access-based cache attacks on AES to practice. In: S&P (2011)
32. Holzer, A., Franz, M., Katzenbeisser, S., Veith, H.: Secure two-party computations in ANSI C. In: CCS (2012)
33. Corporation, Intel: Intel® 64 and IA-32 architectures optimization reference manual. Order Number **248966-032** (2016)
34. Järvinen, K., Kolesnikov, V., Sadeghi, A., Schneider, T.: Garbled circuits for leakage-resilience: hardware implementation and evaluation of one-time programs. In: CHES (2010)
35. Käsper, E., Schwabe, P.: Faster and timing-attack resistant AES-GCM. In: CHES (2009)
36. Kim, T., Peinado, M., Mainar-Ruiz, G.: STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud. In: USENIX Security (2012)
37. Köpf, B., Mantel, H.: Transformational typing and unification for automatically correcting insecure programs. IJIS **6**(2-3) (2007)
38. Köpf, B., Mauborgne, L., Ochoa, M.: Automatic quantification of cache side-channels. In: CAV (2012)
39. Köpf, B., Smith, G.: Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In: CSF (2010)
40. Kreuter, B., Shelat, A., Mood, B., Butler, K.R.B.: PCF: a portable circuit format for scalable two-party secure computation. In: USENIX Security (2013)
41. libtom projects: LibTomCrypt (Version 1.18.2) (2018). <https://github.com/libtom/libtomcrypt/releases/tag/v1.18.2>
42. Malacaria, P., Khouzani, M., Pasareanu, C.S., Phan, Q., Luckow, K.S.: Symbolic side-channel analysis for probabilistic programs. In: CSF (2018)
43. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: USENIX Security (2004)
44. Mantel, H., Schickel, J., Weber, A., Weber, F.: How secure is green it? the case of software-based energy side channels. In: ESORICS (2018)
45. Mantel, H., Starostin, A.: Transforming out timing leaks, more or less. In: ESORICS (2015)
46. Mantel, H., Weber, A., Köpf, B.: A systematic study of cache side channels across AES implementations. In: ESSoS (2017)
47. Matsui, M., Nakaajima, J.: On the power of bitslice implementation on Intel Core2 processor. In: CHES (2007)
48. Mercadier, D., Dagand, P.: Usuba: high-throughput and constant-time ciphers, by construction. In: PLDI, pp. 157–173 (2019)
49. Möller, N.: Nettle (Version 3.5) (2019). <https://ftp.gnu.org/gnu/nettle/nettle-3.5.tar.gz>
50. Molnar, D., Piotrowski, M., Schultz, D., Wagner, D.: The program counter security model: automatic detection and removal of control-flow side channel attacks. In: ICISC (2006)
51. Nane, R., et al.: A survey and evaluation of FPGA high-level synthesis tools. IEEE Trans. CAD Integrat. Circ. Syst. **35**(10), 1591–1604 (2016)
52. National Institute of Standards and Technology: FIPS PUB 46-3: Data encryption standard (DES) (1999)
53. National Institute of Standards and Technology: FIPS PUB 197: Advanced encryption standard (AES) (2001)

54. National Institute of Standards and Technology: Update to current use and deprecation of TDEA (2017). <https://csrc.nist.gov/News/2017/Update-to-Current-Use-and-Deprecation-of-TDEA>
55. OpenSSL Software Foundation: OpenSSL (Version 1.0.1d) (2020). <https://www.openssl.org/source/openssl-1.0.1d.tar.gz>
56. OpenVPN Inc: OpenVPN (2020). <https://openvpn.net/>
57. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of AES. In: CT-RSA (2006)
58. Page, D.: Theoretical use of cache memory as a cryptanalytic side-channel. ePrint 2002/169 (2002)
59. Poddar, R., Datta, A., Rebeiro, C.: A cache trace attack on Camellia. In: InfoSecHiComNet (2011)
60. Smith, G.: On the foundations of quantitative information flow. In: FoSSaCS (2009)
61. Songhori, E.M., Hussain, S.U., Sadeghi, A., Schneider, T., Koushanfar, F.: Tinygarble: highly compressed and scalable sequential garbled circuits. In: S&P (2015)
62. Synopsis: DC Ultra (2020). <https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html>
63. Testa, E., Soeken, M., Amarù, L.G., Micheli, G.D.: Reducing the multiplicative complexity in logic networks for cryptography and security applications. In: DAC (2019)
64. Testa, E., Soeken, M., Riener, H., Amaru, L., Micheli, G.D.: A logic synthesis toolbox for reducing the multiplicative complexity in logic networks. In: DATE (2020)
65. The cURL Team: cURL (2020). <https://curl.haxx.se/>
66. Tillich, S., Smart, N.: (Bristol Format) Circuits of basic functions suitable for MPC and FHE (2020). <https://homes.esat.kuleuven.be/~nsmart/MPC/old-circuits.html>
67. Tsunoo, Y., Saito, T., Suzaki, T., Shigeri, M., Miyauchi, H.: Cryptanalysis of DES implemented on computers with cache. In: CHES (2003)
68. Wang, S., Wang, P., Liu, X., Zhang, D., Wu, D.: Cached: identifying cache-based timing channels in production software. In: USENIX Security (2017)
69. Weiser, S., Spreitzer, R., Bodner, L.: Single trace attack against RSA key generation in Intel SGX SSL. In: ASIACCS (2018)
70. Weiser, S., Zankl, A., Spreitzer, R., Miller, K., Mangard, S., Sigl, G.: DATA - differential address trace analysis: Finding address-based side-channels in binaries. In: USENIX Security (2018)
71. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: FOCS (1986)
72. Zahur, S., Evans, D.: Obliv-C: a language for extensible data-oblivious computation. ePrint 2015/1153 (2015)
73. Zhao, X., Wang, T., Zheng, Y.: Cache timing attacks on Camellia block cipher. ePrint 2009/354 (2009)



Assembly or Optimized C for Lightweight Cryptography on RISC-V?

Fabio Campos¹(✉), Lars Jellema², Mauk Lemmen², Lars Müller¹,
Amber Sprenkels², and Benoit Viguier²

¹ RheinMain University of Applied Sciences, Wiesbaden, Germany
campos@sopmac.de, mail@lars-mueller.com

² ICIS, Radboud University, Nijmegen, The Netherlands
lars.jellema@gmail.com, M.Lemmen@student.ru.nl, amber@electricdusk.com,
b.viguier@cs.ru.nl

Abstract. A major challenge when applying cryptography on constrained environments is the trade-off between performance and security. In this work, we analyzed different strategies for the optimization of several candidates of NIST’s lightweight cryptography standardization project on a RISC-V architecture. In particular, we studied the general impact of optimizing symmetric-key algorithms in assembly and in plain C. Furthermore, we present optimized implementations, achieving a speed-up of up to 81% over available implementations, and discuss general implementation strategies.

Keywords: RISC-V · Lightweight cryptography · Software optimization · NIST

1 Introduction

The enormous growth of the “Internet of Things” (IoT) is changing the world. Forecasts [27] project the number of interconnected embedded devices to around 50 billion worldwide by 2030, a five-fold increase in the next ten years. Driven by the lack of cryptographic algorithms which are more suitable for such constrained environments, NIST started in 2015 a project¹ (NIST-LWC) to solicit, evaluate, and eventually standardize lightweight authenticated encryption algorithms with associated data (AEAD) and hashing. In August 2019, NIST selected 32 candidates for round 2, which is expected to last one year. Lightweight cryptography (LWC), a sub-field of cryptography, covers cryptographic algorithms

Author list in alphabetical order; see <https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf>. This work was supported in part by Continental AG; Elektrobit Automotive GmbH; and the European Commission through the ERC Starting Grant 805031 (EPOQUE).

The full version of this paper is available at <https://ia.cr/2020/836>.

¹ <https://csrc.nist.gov/Projects/lightweight-cryptography>.

intended for use in constrained hardware and software environments. The main goal of NIST's project is to provide algorithms that are more suitable for use on constrained devices where the performance of current NIST cryptographic standards is not acceptable. Thereby, performance figures should be considered on a wide range of 8-bit, 16-bit and 32-bit microcontroller architectures.

On the hardware side, we are facing challenges where critical vulnerabilities [24, 26] cannot be tracked back due to the lack of transparency. The RISC-V project, with roots in academia and research (University of California, Berkeley), has initiated a fundamental shift in the technical and business models for microprocessors. RISC-V [33], a royalty-free and open-source reduced instruction-set architecture (ISA), provides a competitive advantage and the required degree of flexibility to develop secure microprocessors with addresses of 32-, 64-, and 128-bits in length.

Contribution of this Paper. This paper aims at comparing optimization at different levels of round-2 NIST lightweight candidates algorithms on a RISC-V architecture. To achieve this, we first present optimized RISC-V implementations of several cryptographic algorithms. Further, we study the impact of implementing these primitives on RISC-V in assembly compared to in C. Based on this, general implementation strategies are derived and discussed.

Related Work. Many aspects regarding the optimization of lightweight cryptographic algorithms have been studied in the literature. In [28], generic security, efficiency, and some considerations for cryptographic design of lightweight constructions were explored. The modular and reusable architecture of RISC-V facilitates a variety of designs for the implementation of accelerators, ranging from loosely [32] to tightly coupled designs [1]. However, only few works focused on the optimization of cryptographic algorithms on the standard RISC-V instruction set. Stoffelen [31] presented the first optimized assembly implementations of AES, CHACHA, and the KECCAK- f [1600] permutation for the RISC-V instruction set. In [29] the 32s round finalists from the NIST-LWC were evaluated on RISC-V without further optimization.

Organization of this Paper. This paper is structured as follows. Section 2 provides background information on the RISC-V 32-bit architecture and instruction set. We also give the necessary background on the platforms used for benchmarking. In Sect. 3, we briefly recall the selected algorithms and present our optimization strategies, before we describe the benchmarking setup and discuss the achieved results in Sect. 4. Finally, in Sect. 5, we conclude the paper.

Availability of Implementations. We place all software and hardware implementations described in this paper into the public domain (<https://github.com/AsmOptC-RiscV/Assembly-Optimized-C-RiscV>).

2 RISC-V

In Sect. 2.1 we describe in more details the RISC-V 32-bit architecture before detailing the associated instruction set (Sect. 2.2). We then discuss different approaches to execute code targeting RISC-V platform (Sect. 2.3).

2.1 Architecture

The RISC-V architecture uses 32-bit registers numbered from `x0` through `x31`. To ease their use, they also have aliases. `zero` (`x0`) is hard-wired to the value 0; `ra` (`x1`) corresponds to the return address; `sp` (`x2`) to the stack pointer; `gp` (`x3`) to the global pointer; `tp` (`x4`) to the thread pointer. `a0-a7` (`x10-x17`) are function arguments with `a0` and `a1` also functioning as return values. `s0-s11` (`x8-x9`, `x18-x27`) are saved registers. Finally, `t0-t6` (`x5-x7`, `x28-x31`) are temporary registers.

The caller has the responsibility for the saved registers `s0-s11` while the callee is able to freely modify the arguments (`a0-a7`) and temporary registers (`t0-t6`).

Excluding the `zero`, `ra`, `sp`, `gp`, and `tp` registers, we are left with 27 freely usable 32-bit registers. This is twice of what is available in the Cortex-M3 and Cortex-M4 architectures; and it enables us to easily take care of register allocation.

2.2 Instruction Set

The RISC-V base instruction set contains a small number of instructions which we briefly describe here.

Bitwise and arithmetic instructions such as `add`, `addi`, `and`, `andi`, `or`, `ori`, `sub`, `xor`, `xori` take three register operands, or if postfixed by `i`, two registers and one 12-bit sign-extended immediate.

We soon notice missing instructions. *e.g.*, `mov rd, rs` is implemented by taking advantage of the zero register as `add rd, zero, rs`. Similarly, the two's complement negation `neg rd, rs` is replaced by `sub rd, zero, rs` and the one's complement negation `not rd, rs` as `xori rd, rs, -1`. Subtract immediate (`subi`) is written as `addi` with a negative immediate.

The base ISA does not provide rotation instructions but logical and arithmetic shifts: `sll`, `slli`, `srl`, `srlr`, `sra`, and `srai`. Those instructions are read as `shift [left|right] [logical|arithmetic]`.

Load of constants is done with two instructions: `lui` and `addi`. Load upper immediate `lui` takes a 20 bit unsigned immediate and places it in the upper 20 bits of the destination register. The lowest 12 bits are filled with zeros.

```
.equ UART_BASE, 0x40003000

    lui a0,      %hi(UART_BASE)
    addi a0, a0, %lo(UART_BASE)
```

In order to `load words`, `half-words (unsigned)`, or `bytes (unsigned)` from memory, the instructions `lw`, `lh`, `lhu`, `lb`, `lbu` are used. Similarly `sw`, `sh`, `shu`, `sb`, `sbu` are available to `store` values into the memory. For example `lw a5, 8(a2)` will load into `a5` the word located at address `a2 + 8`. Note that the offset has to be a constant. Additionally loads and stores of words have to be 32-bit aligned, *e.g.*, `lw a5, 3(a2)` will fail.

Text labels are used as targets for branches, unconditional jumps and symbol offsets. They are added to the symbol table of the compiled module. Numeric labels are used for local references. When used in jumps and similar instructions, they are suffixed with ‘f’ for a forward reference or ‘b’ for a backwards reference.

```

loop:                                     1:
    ...
    j loop                               j 1b
    j func                               j 2f
    ...
fun:                                     2:
    ...

```

In addition to the `jal` and `jalr` unconditional jump –relative to the program counter or as an absolute address in a register– the instruction `beq`, `blt`, `bltu`, `bge`, `bgeu` are used for conditional jumps. They take three arguments, the first two are used in the comparison while the third one is the destination –label–encoded later as an offset relative to the program counter.

To perform our benchmarks we use the `csrr` instruction (control and status register) to read the 64-bit cycle-counter. On the RV32I architecture, it is split into two 32-bit words (`cycle` and `cycleh`).

2.3 Executing Code

To write optimized code for a specific architecture, we need ways to measure improvements or regressions. Below, we describe 3 test platforms which allowed us to benchmark our code.

SiFive E31 Core. We use a HiFive1 development board. They are easily available and contain the FE310-G000 SoC with an E31 core. The CPU implements the RV32IMAC instruction set. This corresponds to the RV32I base ISA with the extensions for multiplications, atomic instructions and compressed instructions.

It has to be noted that RISC-V does not specify how many cycles an instruction may take or the kind of memory used. As a result benchmarks between different RISC-V cores have to be carefully compared.

The E31 runs at 320+ MHz and uses a 5-stage single-issue in-order pipeline. Additionally it uses a 16 KB, 2-way instruction cache. Fetching an instruction from the cache takes only 1 cycle. Most instruction execution takes 1 cycle with a few exceptions. For example, if there is a cache hit, load word (`lw`) takes 2 cycles, loads of half word (`lh`) and bytes (`lb`) a 3-cycle latency. In the case of a cache miss, the latency is highly dependent on the flash controller’s clock frequency. To prevent such unpredictability, we fill up the cache before any benchmarks.

The E31 comes with a 1-cycle latency branch predictor. It uses a 28-entry branch target buffer (BTB), a 512-entry branch history table (BHT) for the direction of conditional branches, and a 6-entry return-address stack (RAS). A correctly predicted control-flow instruction results in no penalty while mispredictions incur a 3-cycle penalty.

The RISC-V specification requires a 64-bit cycle counter accessible via two CSR registers which we will use to benchmark code. Occasionally measurements may end up taking much longer than expected, we ignore these odd values.

VexRiscv Simulator. VexRiscv is a 32-bit RISC-V CPU implementation written in SpinalHDL. Although it is possible to load the core onto an FPGA; we use the Verilator simulator to emulate a core and flash binaries to it. This process allows us to have cycle counts and to evaluate how each algorithm is performing.

The core features the RV32IM instruction set. This corresponds to the base ISA with the extension for multiplications. We initialize the simulator with 256 KiB of RAM and 128 KB of sRAM.

Similarly to the E31 core, the VexRiscv makes use of a 5-stage pipeline. The absence of a branch predictor and an instruction cache give a significant advantage to algorithms which have been unrolled either by hand or the compiler. This explains major cycle-counts differences in the execution of different implementations of a same algorithm.

riscvOVPSim Simulator. Finally, as opposed to executing code on a board or on a fully simulated core, we use the Open Virtual Platforms developed by Imperas Software, Ltd. Their RISC-V simulator uses Just-in-Time Code Morphing and executes RISC-V code on a Linux or Windows host computer.

This simulator implements the full Instruction Set and permits us to enable or disable specific extensions such as Vector instructions or Bit manipulations. The B extension gives us access to more advanced instructions such as rotations (`rori`, `roli`), packing (`pack`, `packu`), and many others.

Unfortunately, this approach simulates neither pipeline nor cache. While it allows us to execute RISC-V binary files, the results may be biased towards some optimization practices, leading to significant differences between implementations as shown in our benchmarks (see Sect. 4).

3 Optimized Algorithms

Optimized cryptographic implementation are usually written directly in assembly with the idea to prevent the compiler from introducing bugs or weaknesses. By making sure we do not branch on secret data and considering the small size of the RISC-V ISA, we trust the compiler to match our implementations.

We call “Optimized C” the translation of an assembly implementation back into C, making use of `uint32_t` such that the C code mimics the assembly instructions. The underlying idea is to have the compiler further optimize our code and take care of the register allocation.

We now describe the algorithms we optimized and some of the implementation strategies we used.

3.1 GIMLI

GIMLI [6] is a lightweight scheme proposed by Bernstein, Kölbl, Lucks, Massolino, Mendel, Nawaz, Schneider, Schwabe, Standaert, Todo, and Viguier. It makes use of a sponge construction and is based on a 384-bit permutation. Its design puts an emphasis on cross-platform performance and simplicity. The code is compact and uses only logical operations and shifts. The absence of additions allows to “interleave” implementations for platform with different register size than 32 bits. An implementation for RISC-V-64 with the B extension would likely be using such strategy.

The Permutation. The 24-round permutation operates on a 384-bit state seen as a 3×4 matrix of 32-bit words. GIMLI works first locally on the four 96-bit columns; and, to ensure diffusion through the full state, a 2-word swap is applied on the upper 128-bit row of the state every 2 rounds. The symmetries in the state are broken by the addition of an incrementing round constant every 4 rounds.

Using a sponge construction [8], the designers created two variations: a hash function GIMLI-HASH and an authenticated cipher GIMLI-CIPHER.

GIMLI-HASH and GIMLI-CIPHER. GIMLI-HASH initializes a 48-byte state to all-zero before reading sequentially through a variable-length input as a series of 16-byte input blocks. Each full 16-byte input block is absorbed into the state. The final non-full (empty or partial) block is padded with a byte $0x01$ before its absorption while a domain separation byte $0x01$ is XORed in the last byte (47^{th}) of the state. The 32-byte digest output is extracted by blocks of 16-bytes. Each absorption or extraction of blocks is interweaved with calls to the GIMLI permutation.

After initializing the state with a nonce and a key, GIMLI-CIPHER processes the additional data in the same way as GIMLI-HASH. The message is processed in a similar fashion with the exception that after each absorption of a block, the modified first 16 bytes of the state are produced as cipher text. Once the last non-full block is processed; the 16-byte authentication tag is generated from the first 16 bytes of the state.

We are able to get speed-ups on both GIMLI-HASH and GIMLI-CIPHER by optimizing the underlying permutation GIMLI. We rescheduled the order of instructions to avoid swap operations.

Bounds and Optimizations. We optimize GIMLI by first having a deeper look at the inner permutation and by computing the lower bound of the number of cycles used. GIMLI’s state representation uses twelve 32-bit words which are easily contained in the 27 general-use 32-bit registers. [6] shows that only 2 additional registers are required in order to compute the column operations; as a result, in a fully unrolled implementation, the only cycles necessary in the computation are the ones required by the logical operations.

GIMLI uses 2 rotations, 6 XORs, 2 ANDs, 1 OR, and 4 shifts. All logical operations have a latency of 1 cycle, except for rotates which have a 3-cycle latency. A column operation requires thus 19 cycles; iterated over 4 columns and 24 rounds, this totals to 1824 cycles.

GIMLI uses 6 constants (loaded in 2 cycles) derived every 4 rounds (an additional 5 cycles) before being XORed into the state (6 XORs, thus 6 cycles). When the permutation is not directly inlined and used as a function, it requires 12 loads and 12 stores to get the state into registers for an additional 48 cycles. Excluding the cycles needed to preserve some of the callee registers, we have a total of 1885 cycles.

As a base line, the reference C code runs at 2178 cycles. By using careful scheduling of the instructions, and using a minimum number of register – saving into the stack only 4 callee–, our assembly implementation runs at 2092 cycles. The Optimized C version runs in 2132 cycles. This timing difference is explained by the compiler’s use of the 12 callee registers, inducing a 40-cycle penalty.

By unrolling in C –the same approach could have been applied in assembly– over 8 rounds and propagating the swapping by renaming variable to avoid move operations, the compiler manages to achieve further speed-ups by getting down to 1900 cycles. Using this last implementation, we get a 19% speed-up for GIMLI-HASH and GIMLI-CIPHER (Table 1).

Table 1. Cycle counts for different GIMLI implementations on the SiFive board; GIMLI-HASH over 128 bytes of data, GIMLI-CIPHER over a 128 bytes message with 128 bytes of associated data. Compiled with Clang-10 and `-O3`

	C-ref	Assembly	Optimized C	8-round Optimized C
GIMLI	2178	2092 (−4%)	2132 (−2%)	1900 (−13%)
GIMLI-HASH	23120	20812 (−10%)	21055 (−9%)	18678 (−19%)
GIMLI-CIPHER	44423	39583 (−10%)	40816 (−8%)	35853 (−19%)

3.2 SPARKLE

SPARKLE [3] is a family of cryptographic permutations based on the block cipher SPARX [20] and designed by Beierle, Biryukov, Cardoso dos Santos, Großschädl, Perrin, Udovenko, Velichkov, and Wang. SCHWAEMM (an AEAD cipher scheme) and ESCH (a hash function) follow a not hermetic design approach, and share SPARKLE as the underlying permutation. The SPARKLE permutation is a classic ARX design, which, unlike most ARX constructions, provides security guarantees with regard to differential and linear cryptanalysis based on the long trail strategy (LTS) [20]. SCHWAEMM and ESCH work on a relatively small state, which is only 256 bits for the most lightweight instance of SCHWAEMM and 384 bits for the lightest variant of ESCH. The biggest possible state size with 512 bits, can be applied by both schemes. Both algorithms employ the sponge construction.

Two instances for hashing were proposed in [3], i.e., ESCH256 and ESCH384, which allow to process messages of arbitrary length and output a digest of 256 bit, and 384 bit, length, respectively. ESCH256, the main instance of ESCH and the one considered in our work, uses the 384-bit SPARKLE permutation and has a claimed security level of 128 bits.

All the four instances for authenticated encryption with associated data proposed in [3], i.e., SCHWAEMM128-128, SCHWAEMM256-128, SCHWAEMM192-192 and SCHWAEMM256-256 use a variation of the BEETLE mode of operation first presented in [15], which in turn is based on the duplexed sponge construction. We focus again on the main version SCHWAEMM256-128, which uses the 384 bit SPARKLE (SPARKLE384) permutation, with a rate of $r = 256$ bit and a capacity of $c = 128$ bit, claiming a security level of 120 bits.

SPARKLE384 requires 50 rotations, 68 XORs, 24 ADDs, and 2 shifts for a single round. With the exception of rotation (3 cycles), all operations have a latency of 1 cycle. Thus, iterated over 7 or 11 rounds this totals to an estimated lower bound of 1708 cycles, and 2684 cycles respectively. For further details, we refer to the specification [3].

Loop Unrolling. Although unrolling the main loop within the SPARKLE permutation over 7 or 11 rounds results in a significant speed-up (see Table 3) when using instruction cache (like the SiFive core used in this work, see Sect. 2.3), this leads to significantly worse results in the case of AEAD (see Table 2).

Round Constants. In this optimization, we speed-up the permutation by increasing the space required. In every round of the permutation, each of the six ARX-boxes uses the same round constant in their computations. The idea is to avoid the loading of the constants for the ARX-boxes in every round by loading and saving these 6 constants in the registers before the transformation. This comes with the cost of dedicating 6 registers to these constants.

This optimization can be applied in the loop as well as in the unrolled variant of the implementation. In the unrolled implementation, we further reduce the loading of round constants, since these 6 constants are also being used as the round constants that are added to the state every round. In the 7-round variant of the permutation, we save the loading of the first 6 round constants and only have to load the 7th constant. In the 11-round variant of the permutation, we only have to load the 8th constant extra. The other three are already loaded because there are only 8 round constants defined and the selection index is calculated modulo 8. In the loop unrolled implementation we reduce the instruction count for 7 rounds by 72 instructions and for 11 rounds by 126 instructions.

Table 2 shows the achieved speed-up for SCHWAEMM256-128, Table 3 presents the achieved results for ESCH256.

Table 2. Cycle counts for different SCHWAEMM256-128 implementations on the SiFive board; encryption over a 128 bytes of message with 128 bytes of associated data.

Platform	Compiler	Opt	Opt. C	looped + round cst ASM	loop-unrolled Opt. C
SiFive	Clang-10	-03	72286	43877 (-40%)	1059813 (+94%)
SiFive	GCC	-03	71271	42634 (-40%)	1790566 (+95%)
riscvOVPSim	Clang-10	-03	20842	20840 ($\pm 0\%$)	20277 (-3%)
riscvOVPSim	GCC	-03	20762	20161 (-2%)	20010 (-3%)
VexRiscv	GCC	-02	25464	27018 (+6%)	24769 (-3%)

Table 3. ESCH256 cycle counts on each platform. The hashing operation hashes 128 bytes of data.

Platform	Compiler	Opt	Opt. C	loop-unrolled Opt. C
SiFive	Clang-10	-03	62734	34664 (−44%)
SiFive	GCC	-03	58193	33331 (−42%)
riscvOVPsim	Clang-10	-03	17439	16552 (−5%)
riscvOVPsim	GCC	-02	17849	17231 (−3%)
VexRiscv	GCC	-02	18874	17753 (−6%)

3.3 SATURNIN

SATURNIN [14] is the NIST lightweight candidate designed by Canteaut, Duval, Leurent, Naya-Plasencia, Perrin, Pornin, and Schrottenloher. By building on top of a 256-bit block cipher with a 256-bit key, they describe three constructions for hashing (SATURNIN-HASH) and authenticated encryption of small (SATURNIN-SHORT) and large data segment (SATURNIN-CIPHER). This last AEAD scheme uses the counter mode and a separate MAC.

We ported to our benchmark platform the reference implementation and both the 32-bit optimized “bs32” and “bs32x” C implementations [14, Section 3.4.2]. The “bs32” and “bs32x” implementations both implement SATURNIN in a $32 \times$ bitsliced fashion. Their difference is that “bs32” bitslices *inside* of blocks, whereas “bs32x” bitslices *across* blocks. When comparing the two bitsliced implementations, “bs32” showed a consistently better performance than the other, albeit sometimes with a small margin. We decided that “bs32” would be the preferred implementation to use on our platforms.

In all the implementations, we tweaked the code to make sure that any constants would be loaded from SRAM, instead of (the relatively slow) SPI flash. This considerably improved the performance of the bitsliced implementations.

In the end, we see that the Optimized C implementation is considerably faster than the reference implementation in terms of performance, with generally a speed-up by a factor of 2. Another interesting property from the results in Tables 4 and 5 is the performance stability of the implementations across compilers. Where the “bs32” performance is very stable—with cycle counts generally varying less than 10%—the performance of the reference implementation varies a lot with different compiler versions. Nonetheless, we see that newer compiler versions seem to produce faster code.

Table 5 illustrates the fact that the greedy unrolling and inlining by GCC with -03 results in major speed-up on simulators. However once tested on a physical device such as the SiFive development board (2.3), this results in a code too large for the 16KB cache, inducing in a slowdown by a factor of 5.

Table 4. SATURNIN-HASH cycle counts on each platform. The hashing operation hashes 128 bytes of data.

Platform	Compiler	Opt	Ref	bs32
SiFive	Clang-10	-03	49433	28199 (−43%)
SiFive	GCC	-03	78110	30321 (−61%)
riscvOVPSim	Clang-10	-03	46946	27070 (−42%)
riscvOVPSim	GCC	-03	76211	29030 (−61%)
VexRiscv	GCC	-02	103325	32169 (−69%)

Table 5. SATURNIN-CIPHER cycle counts on each platform. The cipher encrypts 128 AD bytes and 128 message bytes.

Platform	Compiler	Opt	Ref	bs32	bs32x
SiFive	Clang-10	-03	121651	59368 (−51%)	68792 (−43%)
SiFive	GCC	-03	151428	60817 (−60%)	5210541 (×34)
SiFive	GCC	-0s	183464	65469 (−64%)	138187 (−24%)
riscvOVPSim	Clang-10	-03	93184	55154 (−41%)	61077 (−34%)
riscvOVPSim	GCC	-03	145734	57366 (−61%)	75646 (−48%)
VexRiscv	GCC	-02	202226	65015 (−68%)	88278 (−56%)

3.4 ASCON

ASCON [21] is a scheme proposed by Dobraunig, Eichlseder, Mendel and Schläpfer. It uses a very small 320-bit state which allows it to fit in registers on most systems. The authors introduce multiple variants of ASCON AEAD as well as a hashing scheme. We focus our efforts on the ASCON-128 AEAD variant. We expect that our results translate fairly well to the other variants and the hashing scheme as they are very similar.

We use the ASCON C [2] repository as a base line, more specifically we use the reference, the 64-bit optimized, and the 32-bit interleaved implementations as starting point for our optimizations.

Improved Formula. First we optimize the inner permutation by improving the ASCON S-box formula (Fig. 1). We reduce the number of required instructions from 22 to 17 and the number of temporary registers from 5 to 3 at the cost of less potential for parallelism. Instruction-level parallelism—such as out-of-order execution—is common in high-end CPUs but not so common in lightweight platforms like our RISC-V targets. This optimization gives us a 10% speed-up for both the assembly and Optimized C implementations (Table 6).

$$\begin{aligned}
o_0 &= i_3 \oplus i_4 \oplus (i_1 \vee (i_0 \oplus i_2 \oplus i_4)) \\
o_1 &= i_0 \oplus i_4 \oplus ((i_1 \oplus i_2) \vee (i_2 \oplus i_3)) \\
o_2 &= i_1 \oplus i_2 \oplus (i_3 \vee \neg i_4) \\
o_3 &= i_1 \oplus i_2 \oplus (i_0 \vee (i_3 \oplus i_4)) \\
o_4 &= i_3 \oplus i_4 \oplus (i_1 \wedge \neg(i_0 \oplus i_4))
\end{aligned}$$

Fig. 1. These formulas compute the Ascon S-box in 17 operations (once duplicate operations are taken out); o_n indicates output bit n and i_n indicates input bit n .

Table 6. Cycle counts for the different ASCON’s round functions over 6 rounds; Compiled with Clang-10 and -O3

Platforms	C-ref	Assembly	Optimized C
SiFive	832	750 (−10%)	750 (−10%)
riscvOVPsim	830	748 (−10%)	748 (−10%)

Bit Interleaving. We also compare the C implementation optimized for 32-bit interleaving. It performs the worst of all others including the baseline implementation. Bit interleaving allows 32-bit rotations to model 64-bit rotations efficiently, unfortunately our targets does not support 32-bit rotations. We expect this implementation will perform better when targeting RISC-V cores comes with the B extension, which adds rotation instructions.

Optimized 64 Bits. Finally, we compare the C implementation optimized for 64-bit processors. On RISC-V cores without the B extension, the 64-bit operations are compiled to 32-bit operations in a straightforward manner and the compiler has no trouble with it. As RISC-V does not support misaligned memory access, we had to modified the code to handle the authentication tag.

While on the RISC-V OVP simulator the 64-bit optimized version is 7% faster than the baseline, testing it on the SiFive board reveals significant slowdowns due to the code not fitting in the 16KB instruction cache.

Our final implementation makes use of the improved S-box formula in a 6-round unrolled Optimized C permutation. By folding the processing of associated data and message we are able to reuse the code and have to compiled code fit in the instruction cache. Applying these modifications, we achieve our best results: 15% faster than the baseline (Table 7).

Table 7. Cycle counts for different ASCON implementations in OVP sim for encrypting 128 bytes of message and 128 bytes of associated data; compiled with Clang-10 and -03

Implementation	OVP sim	SiFive
ref. & default permutation	31990	32038
ref. & asm permutation	28988 (−9%)	29036 (−9%)
ref. & inlined Optimized C perm	27489 (−14%)	27703 (−14%)
bit interleaved inline permutation	32001 (±0%)	1559691 (×49)
opt. 64-bit & default unrolled perm	29646 (−7%)	1191702 (×37)
opt. 64-bit & asm permutation	29090 (−9%)	29170 (−9%)
opt. 64-bit & fully unrolled Opt. C perm	27589 (−14%)	809631 (×25)
opt. 64-bit & 6-round unrolled Opt. C perm	27184 (−15%)	27271 (−15%)

3.5 Delirium

ELEPHANT [11] is a family of lightweight authenticated encryption schemes designed by Beyne, Chen, Dobraunig, and Mennink. The mode of ELEPHANT is a nonce-based encrypt-then-MAC construction, where encryption is performed using counter mode based on permutation masked using LFSRs. One of the instances of ELEPHANT is ELEPHANT-KECCAK- f [200], also called DELIRIUM, which uses KECCAK as its permutation primitive. DELIRIUM has a state size of 200 bits and claimed a security level of 127 bits. We optimize DELIRIUM by exploiting ELEPHANT’s possibility for parallelization by using bit-interleaving.

Bit Interleaving. In order to make full use of the 32-bit registers, we combine four blocks of byte-sized elements into one block of 4-byte elements. Thus, we can process four blocks at the same time and our state representation changes to an array of 25 32-bit words (5-by-5-by-32) with a total size of 800 bits. In this new representation, one block amounts to four blocks in the standard representation.

There are two possible cases when transforming blocks before encrypting/decrypting to the new representation. The first and the easiest case is when the amount of blocks that need to be transformed is a multiple of four. This means that all groups of four blocks consisting of 8-bit words can be interleaved to make one block of 32-bit words. The second case is when the amount of blocks is not a multiple of four. Since the new representation needs four “old” blocks to transform into one new block, we have to use padding blocks filled with zero values to add to make the amount of blocks to a multiple of four.

After encryption/decryption, when transforming back to a byte representation of the data, we have to de-interleave each interleaved 32-bit block back to four blocks of bytes. Since it is possible that the amount of original blocks was not a multiple of four, we need to make sure none of the data from the added padding blocks gets joined in the output data. This can be done by cutting off any output data which exceeds the message-length variable.

As shown in Table 8, we note that shorter inputs perform worse in the optimized implementation. This is because the effort of interleaving data to process four blocks simultaneously is wasted if there are very few blocks to process.

Table 8. Cycle counts for different ELEPHANT-KECCAK- $f[200]$ implementations on the SiFive board; encryption over a 32/64/128 bytes message with 32/64/128 bytes of associated data.

Platform	Compiler	Message length	Data length	C-ref	Bit interleaved
SiFive	GCC	16	16	66541	73989 (+11%)
SiFive	GCC	32	32	91837	74385 (−19%)
SiFive	GCC	64	64	143181	74890 (−47%)
SiFive	GCC	128	128	245100	113031 (−53%)
SiFive	Clang-10	128	128	241975	145936 (−40%)
riscvOVPsim	GCC	32	32	64651	66690 (+3%)
riscvOVPsim	GCC	64	64	102138	66805 (−35%)
riscvOVPsim	GCC	128	128	176086	101966 (−42%)
riscvOVPsim	Clang-10	128	128	163973	103631 (−37%)

3.6 XOODYAK

XOODYAK [18]—designed by Daemen, Hoffert, Peeters, Van Assche, and Van Keer—based on the XOODOO permutation [16, 17], is a cryptographic scheme that is suitable for several symmetric-key functions, including hashing, encryption, MAC computation and authenticated encryption. XOODOO, according to its authors [17], can be seen as a porting of the KECCAK- p [9, 10] design approach to a GIMLI-shaped [6] state.

XOODOO iteratively applies 12 rounds to a 384-bit state, which can be treated as 3 horizontal planes, each one consisting of 4 parallel 32-bit lanes. The choice of 12 rounds justifies a security claim in the hermetic philosophy. The claimed security strength for XOODYAK is 128 bits.

An estimated lower bound for cycles taken by XOODOO can be calculated as follows. It requires 24 rotations, 37 XORs, 12 ANDs, and 12 NOTs for a single round. With the exception of rotation (3 cycles), all operations take 1 cycle. Thus, iterated over 12 rounds this totals to 1596 cycles.

Lane Complementing. The idea behind lane complementing, first proposed in the KECCAK implementation overview [10], is to reduce the number of NOT instructions by complementing certain lanes before the transformation.

In XOODOO the state is ordered in 4 sheets, each containing 3 lanes with a width of 32-bit. The χ layer computes 3 XOR, 3 AND and 3 NOT operations for every sheet in the state. This sums up to 12 NOT operations per round and 144

NOT operations in total. In the default case, the χ transformation for every lane $a[i]$ in a sheet, with $0 \leq i \leq 2$ and index calculation mod 3, can be calculated as shown in Eq. (1).

$$a[i] \leftarrow a[i] \oplus (\overline{a[i+1]} \wedge a[i+2]) \quad (1)$$

For example, we now want to complement lane $a[2]$. Thus, the equation of lane $a[0]$ gets rearranged as follows:

$$\begin{aligned} a[0]' &= a[0] \oplus (\overline{a[1]} \wedge \overline{\overline{a[2]}}) = a[0] \oplus (\overline{a[1]} \wedge \overline{a[2]}) = \overline{a[0] \oplus (a[1] \vee \overline{a[2]})}, \\ \overline{a[0]'} &= a[0] \oplus (a[1] \vee \overline{a[2]}). \end{aligned}$$

The complementation of $a[2]$ results in the cancellation of the negation of $a[1]$, the switch from an AND to an OR operation and the complement of $a[0]'$. Now we calculate all three lanes of a sheet with the complement of the lane $a[2] \leftarrow \overline{a[2]}$:

$$\begin{aligned} a[0] &\leftarrow \overline{a[0]'} = a[0] \oplus (a[1] \vee a[2]), \\ a[1] &\leftarrow a[1]' = a[1] \oplus (a[2] \wedge \overline{a[0]}), \\ a[2] &\leftarrow \overline{a[2]'} = a[2] \oplus (a[0] \wedge a[1]). \end{aligned}$$

It can be observed that we only need one complementation for this sheet, instead of three. For the computation of $a[1]$, $a[0]$ is complemented to be positive, because $a[0]$ was negated before. This example of lane complementing comes with the cost of applying the input mask $\overline{a[2]}$ and output mask $\overline{a[0]}$, $\overline{a[2]}$.

The possible transformations of the boolean equations for a sheet are not fixed to one. Thus, there are multiple boolean equations that are still logically congruent, but may differ in the input and output mask. We want to find the boolean equations and input mask with the lowest possible number of NOT-instructions. To simplify this problem, we set the boolean equations to a fixed set and only care about the possible input patterns. Therefore, we employ an algorithm for finding the minimum NOT instruction count for a certain set of boolean equations. We test all 2^{12} possible combinations of input masks. For every input mask, we follow the complements propagation through the 12 rounds of the permutation as a symmetric difference pattern in the state and count the NOT instructions.

After the application of the algorithm, we obtain an input mask and a sequence of boolean equations. This input mask is 2-round invariant, meaning that the input mask is the always same after every two rounds. Hence, it can be implemented as a loop and therefore have a smaller code size.

We reduce the number of NOT operations to exactly 33% over 12 rounds. The application of our input and output mask, each costs 4 NOT operations. Due to a larger number of lanes in KECCAK, Stoffelen [31] achieved a reduction to 20%.

Lane complementing is not an assembly-specific optimization. As shown in Table 9 and 11, we achieve a very similar speed-up in assembly and in C (Table 10).

Table 9. Cycle counts for different implementations of XOODYAK in AEAD mode GCC compiled with `-O2` in riscvOVPSim for encrypting 128 bytes of message and 128 bytes of associated data.

Implementation	riscvOVPSim	Relative
Reference	105463	
Loop unrolled + lane complementing assembly	29574	-71%
Loop unrolled + lane complementing Optimized C	28672	-72%

Table 10. Cycle counts for XOODYAK in hash mode on each platform, compiled with `-O3`. The hashing operation hashes 128 bytes of data.

Platform	Compiler	Ref	Unrolled & lane comp
SiFive	Clang-10	81349	17963 (-78%)
SiFive	GCC	82741	17063 (-79%)
riscvOVPSim	Clang-10	18114	16845 (-7%)
riscvOVPSim	GCC	23247	16614 (-29%)
VexRiscv	GCC <code>-O2</code>	261678	38378 (-85%)

Table 11. Cycle counts for XOODYAK in AEAD mode on each platform, compiled with `-O3`, for encrypting 128 bytes of message and 128 bytes of associated data.

Platform	Compiler	Ref	Unrolled & lane comp
SiFive	Clang-10	103717	26246 (-75%)
SiFive	GCC	103522	23238 (-78%)
riscvOVPSim	Clang-10	25002	23429 (-6%)
riscvOVPSim	GCC	29775	21668 (-28%)
VexRiscv	GCC <code>-O2</code>	261678	38378 (-85%)

3.7 AES

In [31], Stoffelen proposes two assembly implementations of AES: the first one is based on lookup tables, and the second one uses a bitsliced approach.

With a Lookup Table. When encrypting a single block of 16 bytes, multiple steps of the round function can be combined in a lookup table, also called T-table by Daemen and Rijmen in [19]. Note that this type of implementation is

Table 12. Cycle counts for the Assembly of [31] and its translation to C on the SiFive board, compiled with Clang-10 and `-O3`.

	Assembly	Optimized C
Key schedule	342	342
1-block encryption	903	901

usually vulnerable to cache attacks [4, 12, 30]. Because none of our benchmarking platforms have a data cache, we believe this implementation is likely “safe” to use.

For his table-based implementation, Stoffelen makes use of the baseline instructions described in [7]. Most of the proposed optimization by Bernstein and Schwabe are not applicable due to the small instruction set of the RISC-V architecture. The translation from assembly to C using `uint32_t` to simulate registers is straightforward, and the lookup table is converted to an array as `uint32_t variable[]` (Table 12).

Note that if the table is declared as `const`, the compiler will place it in the `.rodata` segment. While this change does not have any impact on the verilator and the riscvOVPSim simulators, it induces a major slowdown in the case of the SiFive board as the SPI flash is significantly slower than the SRAM.

In order to prevent the compiler from messing with the pointer arithmetic, data pointers are kept in the `uint8_t*` type. This forces us to cast the pointer to `uint32_t*` before de-referencing to trigger the compiler to use the `lw` instruction.

```
Y0 = RK[0]; T0 = (uint32_t*)(LUT1 + ((*X0 & 0xff) << 4)); Y0 = Y0 ^ *T0;
Y1 = RK[1]; T1 = (uint32_t*)(LUT1 + ((*X1 & 0xff) << 4)); Y1 = Y1 ^ *T1;
Y2 = RK[2]; T2 = (uint32_t*)(LUT1 + ((*X2 & 0xff) << 4)); Y2 = Y2 ^ *T2;
Y3 = RK[3]; T3 = (uint32_t*)(LUT1 + ((*X3 & 0xff) << 4)); Y3 = Y3 ^ *T3;
```

Listing 1.1. Code fragment of AES encryption

Using a Bitsliced Approach. When using AES in CTR or GCM mode, multiple blocks can be processed in parallel using a bitsliced implementation [23, 25]. This strategy is often more efficient and avoids lookup tables, making the implementation more resistant against timing attacks.

By using the same approach as with lookup tables, we translate the assembly from [31] back into C. As seen in Table 13 the key schedule it is slightly slower. However this translation approach gives us a 4% speed-up in the case of the encryption in CTR mode (Table 13).

Table 13. Cycle counts for the Assembly of [31] and its translation to C on the SiFive board, compiled with Clang-10 -O3.

	Assembly	Optimized C
Key schedule	1248	1256
Encryption of 128 blocks	260695	249813 (−4%)

3.8 Keccak

We now have a look at the KECCAK- f family permutation—designed by Bertoni, Daemen, Peeters and Van Assche [9]—, more precisely its 1600-bit instance found in the SHA-3 standard by NIST[22]. The permutation is used in multiple cryptographic constructions including future post-quantum candidates such as

FrodoKEM[13], SPHINCS+[5] and others. Stoffelen [31] provides us with another optimized implementation for RISC-V inspired by the *Keccak implementation overview*[10]. KECCAK- f [1600] works on a state composed of 25 64-bit lanes, in other words a total of 50 32-bits words. This is more than the number of register made available by the ISA, preventing the state from completely fitting in the registers. By using bit interleaving and other techniques, Stoffelen manages to reduce the number of cycles used.

Table 14. Cycle counts for the Assembly of Keccak [31] and its translation to C on the SiFive board, compiled with GCC `-Os`.

	Assembly	Optimized C
KECCAK- f [1600]	13731	13336 (−3%)

We take his implementation and translate it back to C. We compile with GCC and `-Os` instead of `-O2` or `-O3` to get slightly faster results than the assembly implementation in [31] (Table 14).

4 Comparison with Other Implementations and Additional Benchmark

Some other implementations of lightweight candidates are publicly available; we chose to compare our work against the repository of Weatherley² as their implemented are “focused on good performance in plain C on 32-bit embedded microprocessors”.

As Clang-10 generally produces faster results than GCC with `-O3`, we used it to compile and benchmark every optimized C implementation provided by Weatherley. We measure the cycle counts for encryption of AEAD schemes for 128-byte messages with 128 bytes of associated data. In Table 15, we summarize the performance of our software and Weatherley’s implementations.

While on the OVP simulator most of our implementations produces just slightly better results with an average at −4% cycle counts; when using the SiFive board, the unrolled implementation of Weatherley suffer heavily from the 16KB instruction cache. This makes our RISC-V-optimized code on average 47.5% faster.

² <https://github.com/rweather/lightweight-crypto>, commit 52c8281.

Table 15. Cycle counts for AEAD mode on the SiFive and riscvOVPSim platform, compiled with Clang-10 -O3, for encrypting 128 bytes of message and 128 bytes of associated data.

Algorithm	Weatherley		Our results	
	OVP	SiFive	OVP	SiFive
GIMLI	37596	38530	35690 (-5%)	35853 (-7%)
SCHWAEMM256-128	20842	72286	20277 (-3%)	43877 (-40%)
SATURNIN	55367	152803	55154 (-1%)	59368 (-61%)
ASCON	41228	42562	27184 (-34%)	27271 (-36%)
DELIRIUM	110171	765235	103631 (-6%)	145936 (-81%)
XOODYAK	18852	64869	23451 (+24%)	26246 (-60 %)

5 Conclusion

We described how multiple lightweight NIST candidates such as GIMLI, SPARKLE, SATURNIN, ASCON, DELIRIUM, and XOODYAK can be efficiently implemented. With strategies such as loop unrolling, we are able to write assembly code close to the lower bound given by the number instructions arithmetic. By translating our assembly implementation back into C, we get the compiler to further optimize our results.

Using the AES and KECCAK assembly implementations from Stoffelen [31], we also show that our approach is applicable to existing code bases, and may provide slightly improved results while increasing the readability and maintainability of the code.

We use the HiFive1 development board to illustrate that algorithms need to be tested on physical devices in order to guarantee useful optimized implementations. Although strategies such as fully unrolled loops may work nicely in simulated environments such as riscvOVPSim; they will fail at length on physical devices with *e.g.*, a 16KB instruction cache.

As the NIST lightweight competition is currently taking place, we hope our results will be found useful by the candidates' implementers and designers. On the other side, RISC-V offers the opportunity to disrupt the processor industry by using a very collaborative approach offering more interoperability and partnership opportunities.

References

1. Alkim, E., Evkan, H., Lahr, N., Niederhagen, R., Petri, R.: ISA extensions for finite field arithmetic: accelerating Kyber and NewHope on RISC-V. IACR Trans. Cryptograph. Hardware Embedded Syst. **2020**(3), 219–242 (2020). <https://tches.iacr.org/index.php/TCHES/article/view/8589>
2. Ascon C repository on GitHub. <https://github.com/ascon/ascon-c>

3. Beierle, C.: Lightweight AEAD and hashing using the sparkle permutation family. *IACR Trans. Symmetr. Cryptol.* **2020**(S1), 208–261 (2020). <https://tosc.iacr.org/index.php/ToSC/article/view/8627>
4. Bernstein, D.J.: Cache-timing attacks on AES (2005). <https://cr.yp.to/antiforgery/cachetiming-20050414.pdf>
5. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS+ signature framework. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, New York, NY, USA, 2019*, pp. 2129–2146. Association for Computing Machinery (2019). <https://doi.org/10.1145/3319535.3363229>
6. Bernstein, D.J., et al.: Gimli: a cross-platform permutation. In: *Cryptographic Hardware and Embedded Systems - CHES 2017* (2017). <https://eprint.iacr.org/2017/630>
7. Bernstein, D.J., Schwabe, P.: New AES software speed records. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) *INDOCRYPT 2008*. LNCS, vol. 5365, pp. 322–336. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89754-5_25
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: *ECRYPT Hash Workshop*, vol. 2007 (2007). <https://keccak.team/files/SpongeFunctions.pdf>
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Johansson, T., Nguyen, P.Q. (eds.) *EUROCRYPT 2013*. LNCS, vol. 7881, pp. 313–314. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_19
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: Keccak implementation overview (2013). <https://keccak.team/files/Keccak-implementation-3.2.pdf>
11. Beyne, T., Chen, Y.L., Dobraunig, C., Mennink, B.: Elephant v. 1 (2019). <https://www.esat.kuleuven.be/cosic/elephant/>
12. Bonneau, J., Mironov, I.: Cache-collision timing attacks against AES. In: Goubin, L., Matsui, M. (eds.) *CHES 2006*. LNCS, vol. 4249, pp. 201–215. Springer, Heidelberg (2006). https://doi.org/10.1007/11894063_16
13. Bos, J., et al.: Frodo: take off the ring! practical, quantum-secure key exchange from LWE. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, New York, NY, USA, 2016*, pp. 1006–1018. Association for Computing Machinery (2016). <https://doi.org/10.1145/2976749.2978425>
14. Canteaut, A., et al.: Saturnin: a suite of lightweight symmetric algorithms for post-quantum security. *IACR Trans. Symmetr. Cryptol.* **2020**(S1), 160–207 (2020). <https://tosc.iacr.org/index.php/ToSC/article/view/8621>
15. Chakraborti, A., Datta, N., Nandi, M., Yasuda, K.: Beetle family of lightweight and secure authenticated encryption ciphers. *IACR Trans. Cryptograph. Hardware Embedded Syst.* **2018**(2), 218–241 (2018). <https://tches.iacr.org/index.php/TCHES/article/view/881>
16. Daemen, J., Hoffert, S., Van Assche, G., Van Keer, R.: The design of Xoodoo and Xooff. *IACR Trans. Symmetr. Cryptol.* **2018**(4), 1–38 (2018). <https://tosc.iacr.org/index.php/ToSC/article/view/7359>
17. Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Xoodoo cookbook. *Cryptology ePrint Archive, Report 2018/767* (2018). <https://eprint.iacr.org/2018/767>
18. Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Xoodyak, a lightweight cryptographic scheme. *IACR Trans. Symmetr. Cryptol.* **2020**(S1), 60–87 (2020). <https://tosc.iacr.org/index.php/ToSC/article/view/8618>

19. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer, Heidelberg (2002). <https://doi.org/10.1007/978-3-662-04722-4>
20. Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Großschädl, J., Biryukov, A.: Design strategies for ARX with provable bounds: SPARX and LAX. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 484–513. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_18
21. Dobraunig, C., Eichlseder, M., Mendel, F., Schl affer, M.: Ascon v1.2 (2016). <https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf>
22. Dworkin, M.J.: FIPS 202: SHA-3 standard: permutation-Based Hash and Extendable-Output Functions. Technical report, National Institute of Standards and Technology (2015). <https://doi.org/10.6028/NIST.FIPS.202>
23. K asper, E., Schwabe, P.: Faster and timing-attack resistant AES-GCM. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 1–17. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04138-9_1
24. Kocher, P., et al.: Spectre attacks: exploiting speculative execution. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 1–19 (2019). <https://ieeexplore.ieee.org/document/8835233>
25. K onighofer, R.: A fast and cache-timing resistant implementation of the AES. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 187–202. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79263-5_12
26. Lipp, M., et al.: Meltdown: reading kernel memory from user space. In 27th USENIX Security Symposium (USENIX Security 2018) (2018). <https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-lipp.pdf>
27. Liu, S.: IoT connected devices worldwide 2030 (2019). <https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/>
28. Mouha, N.: The design space of lightweight cryptography. In: NIST Lightweight Cryptography Workshop 2015, Gaithersburg, United States, July 2015. <https://hal.inria.fr/hal-01241013/file/session5-mouha-paper.pdf>
29. Nisanci, G., Atay, R., Pehlivanoglu, M.K., Kavun, E.B., Yalcin, T.: Will the future lightweight standard be RISC-V friendly? (2019). <https://csrc.nist.gov/CSRC/media/Presentations/will-the-future-lightweight-standard-be-risc-v-fri/images-media/session4-yalcin-will-future-lw-standard-be-risc-v-friendly.pdf>
30. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006). https://doi.org/10.1007/11605805_1
31. Stoffelen, K.: Efficient cryptography on the RISC-V architecture. In: Schwabe, P., Th eriault, N. (eds.) LATINCRYPT 2019. LNCS, vol. 11774, pp. 323–340. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30530-7_16
32. Wang, W., et al.: XMSS and embedded systems. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 523–550. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38471-5_21
33. Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanovi. The RISC-V Instruction Set Manual. Volume 1: User-Level ISA, Version 2.2, 2017

Codes and Lattices



Attack on LAC Key Exchange in Misuse Situation

Aurélien Greuet¹, Simon Montoya^{1,2(✉)}, and Guénaél Renault²

¹ IDEMIA France, Paris La Défense, France

{aurelien.greuet,simon.montoya}@idemia.com

² LIX, INRIA, CNRS, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France

guenael.renault@lix.polytechnique.fr

Abstract. LAC is a Ring Learning With Error based cryptosystem that has been proposed to the NIST call for post-quantum standardization and passed the first round of the submission process. The particularity of LAC is to use an error-correction code ensuring a high security level with small key sizes and small ciphertext sizes. LAC team proposes a CPA secure cryptosystem, LAC.CPA, and a CCA secure one, LAC.CCA, obtained by applying the Fujisaki-Okamoto transformation on LAC.CPA. In this paper, we study the security of LAC Key Exchange (KE) mechanism, using LAC.CPA, in a misuse context: when the same secret key is reused for several key exchanges and an active adversary has access to a *mismatch oracle*. This oracle indicates information on the possible mismatch at the end of the KE protocol. In this context, we show that an attacker needs at most 8 queries to the oracle to retrieve one coefficient of a static secret key. This result has been experimentally confirmed using the reference and optimized implementations of LAC. Since our attack can break the CPA version in a misuse context, the Authenticated KE protocol, based on the CCA version, is not impacted. However, this research provides a tight estimation of LAC resilience against this type of attacks.

1 Introduction

The threat of a quantum computer that breaks most of the current public-key cryptosystems with Shor's Algorithm [18], led the National Institute of Standards and Technology (NIST), in 2016, to begin a call for post-quantum safe public-key cryptography [15]. The NIST specifically asked for quantum safe Key Encapsulation Mechanisms (KEMs).

Among the different quantum resistant cryptosystems, those using ideal lattices based on a Ring instantiation of the Learning With Errors problem (RLWE) [12] are believed to be a promising direction to provide efficient and secure candidates. Indeed, 4 out of the 17 remaining KEMs of the round 2 of the NIST submissions are ideal lattices based on the RLWE problem [1, 2, 9, 19]. The interest of RLWE based KEM is confirmed by real life experiments. In 2016, Google

started to experiment RLWE based KEM between Chrome and Google’s services. Moreover, several RLWE-based KEMs are implemented by the Open Quantum Safe project in their OpenSSL and OpenSSH forks. This project involves academics, like University of Waterloo, and technology companies like Amazon Web Services or Microsoft Research. However, before a world-wide practical deployment of lattice-based KEMs, it is interesting to assess their security in different scenarios, for example in misuses conditions.

Motivation

In this paper we study LAC [19], a RLWE candidate to the NIST standardization process. It differs from other RLWE KEMs by its small key and ciphertext sizes, for an equivalent security level. Such small sizes can be an advantage, particularly in constrained environments and embedded systems. We focus on LAC.KE, a KEM based on the CPA secure public-key cryptosystem LAC.CPA. In constrained environments it’s interesting to determine the impact of key caching to evaluate the requirement of random generation. Furthermore, the specification of CCA version of LAC uses a static secret key due to security provided by the Fujisaki Okamoto (FO) transformation [7]. However, as shown in [4, 16] without a secure implementation of FO transformation, a physical attack can bypass security provided by FO and modifies a CCA version to a CPA one with a static secret key.

Our study is inspired by previous works in [4, 11, 17], which evaluate the resilience in a misuse context offered by two other NIST KEM candidates. Here we propose to pursue this evaluation with another NIST candidate to determine which one is the more resilient against this kind of attack.

Previous Works

The seminal work of Menezes and Ustaoglu [13] paved the way for active attacks on KE protocols. The idea of key mismatch attack on LWE based key exchange was first proposed by Fluhrer in [5, 6]. In a key mismatch attack, a participant’s secret key is reused for several key establishments, and his private key can be recovered by comparing the shared secret key of the two participants.

Some lattice-based KEM of the NIST competition were analysed in the key reused context using a key mismatch oracle. In [3], Baetu *et al.* proposed a generic attack for several algorithms using the same structure called meta-algorithm. However, most of the algorithms attacked in [3] did not pass the first round of the submission, except Frodo-640 and NewHope512. However in [10], Huguenin-Dumittan *et al.* pursue the work of generic attack for round 2 candidates. The security of NewHope1024 CPA algorithm in this misuse scenario is analyzed by Bauer *et al.* in [4] and an improvement is proposed in [11]. More recently, in the same context, an attack on Kyber CPA KEM is proposed by Ding *et al.* [17].

In [8], Guo *et al.* presented an attack against the CCA version of LAC. This attack is theoretically stronger than ours since it does not rely on a misuse hypothesis but it requires 2^{162} pre-computations that cannot be achieved in practice.

Our Contribution

In this article, we investigate the resilience of the LAC KEM under a misuse case: we assume that the same secret key is reused for multiple key establishment and we assume that an attacker can use a key mismatch oracle as introduced in [4].

Since LAC uses encoding and compression functions different from a classical RLWE scheme, Fluhrer’s attack [6] cannot be applied directly. Furthermore, these functions are different from those used in NewHope or Kyber, so we cannot apply straightforwardly the attacks described in [4, 11, 17]. A recent independent work in [10] attacks several round 2 candidates using the generic structure of these schemes. Their attack is applied to the first security level of LAC but is focused on the theoretical aspect. Our work complete this work by bringing a practical aspect and an extension to the others security levels.

The main idea of these attacks is to send forged ciphertexts to a victim, ensuring that its decryption will leak partial information of his static secret key. LAC algorithms use two encoding functions including an error-correction code BCH that can correct a limited number of errors. If a message exceeds the number of errors that the error-correction code can correct, then a decryption failure occurs. Thus, we propose to use this failure to provide leaks about the static secret key.

More precisely, we propose a deterministic key mismatch attack on LAC KE for the first two security levels: LAC-128 and LAC-192, which required at most 2 queries per coefficient of the secret key. Afterwards, we adapt our attack to the highest security level LAC-256 which is still deterministic but we need at most 8 queries per coefficient of the secret key.

We experimented our attack with the reference and optimized implementation in C provided by the LAC team [19] with parameters described in Sect. 2.2. The code of our attack is available in [14].

Organization

In Sect. 2, we introduce some notations, describe LAC.CPA and LAC Key Exchange Mechanism and present the different parameters used in LAC algorithms. In Sect. 3, we describe the notion of *key mismatch oracle* introduced in [4] and the attack for the first two security levels. Finally, in Sect. 4 we adapt the attack to the higher security level.

2 Preliminaries

2.1 Notation

Ring Definition. For an integer $q \geq 1$, let \mathbb{Z}_q be the residue class group modulo q such that \mathbb{Z}_q can be represented as $\{0, \dots, q-1\}$. We define R_q being the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

Polynomial. A polynomial in R_q is of degree at most $(n - 1)$ with coefficients in \mathbb{Z}_q . Given $P \in R_q$, we denote by $P[i]$ or P_i the coefficient associated with the monomial x^i . P can also be represented as a vector with n coordinates. In the following, the notation $(a)_{l_v}$ ($l_v \in \mathbb{N}$), where a is a vector (or a polynomial) of dimension $n > l_v$, means we keep the first l_v coordinates of a .

Message Space. Let the message space \mathcal{M} be $\{0, 1\}^{l_m}$ and the space of random seeds \mathcal{S} be $\{0, 1\}^{l_s}$, where l_m and l_s are two integer values.

Random Distribution. Let ψ_σ be the centred binomial distribution on the set $\{-1, 0, 1\}$. We denote the centred binomial distribution for n independent coordinates by ψ_σ^n i.e. for a vector a of dimension n each coefficient is sampled with the centred binomial distribution. In LAC algorithms we use:

1. $\psi_1 : Pr(x = 0) = \frac{1}{2}, Pr(x = -1) = \frac{1}{4}, Pr(x = 1) = \frac{1}{4}$
2. $\psi_{\frac{1}{2}} : Pr(x = 0) = \frac{3}{4}, Pr(x = -1) = \frac{1}{8}, Pr(x = 1) = \frac{1}{8}$

Given a set A , $U(A)$ is the uniform distribution over A . We denote by H a hash function and **Samp**($D, seed$) an algorithm which samples a random variable according to a distribution D with a given seed.

Error Correction Code. We denote by $[n', k, d]$ a set of parameters of an error-correction code (in our case a binary BCH code). n' denotes the length of the codewords, k is the dimension and d is the minimal Hamming distance of the code.

2.2 LAC

LAC is a Ring-LWE based public key encryption scheme over R_q . In order to balance performance and size, LAC team chose $q = 251$, that fits on one byte. This choice of a small modulus implies a lower security or a higher decryption error rate. To overcome these issues, an error-correction code is used, allowing to keep a low decryption error rate and maintain the same security level than schemes using larger modulus. Three security levels are proposed for LAC: LAC-128, LAC-192 and LAC-256. In this section, we describe the four algorithms **CPA.KeyGen**, **CPA.Encrypt**, **CPA.Decrypt**, **CPA.Decrypt256** of the CPA version of LAC, the four subroutines **BCHEncode**, **BCHDecode**, **Compress** and **Decompress** and the CPA-KEM scheme.

Note that **KeyGen** and **Encrypt** are common to the three security levels. However, the decryption depends on the security level: Algorithm 3 is the decryption process for LAC-128 and LAC-192. The decryption routine for LAC-256 is described in Algorithm 4.

Algorithm 1.*CPA.KeyGen()*

Ensure: Key pair (p_k, s_k)

- 1: $seed_a \leftarrow U(\mathcal{S})$
- 2: $a \leftarrow \mathbf{Samp}(U(R_q), seed_a) \in R_q$
- 3: $s \leftarrow \psi_\sigma^n$
- 4: $e \leftarrow \psi_\sigma^n$
- 5: $b \leftarrow a \times s + e \in R_q$
- 6: **return** $(p_k, s_k) = ((seed_a, b), s)$

Algorithm 2.*CPA.Encrypt* $(p_k, m, seed)$

Ensure: Ciphertext $c = (c_1, c_2)$

- 1: $(seed_a, b) \leftarrow p_k$
- 2: $a \leftarrow \mathbf{Samp}(U(R_q), seed_a) \in R_q$
- 3: $\hat{m} \leftarrow \mathbf{BCHEncode}(m) \in \{0, 1\}^{l_v}$
- 4: $r \leftarrow \mathbf{Samp}(\psi_\sigma^n, seed)$
- 5: $e_1 \leftarrow \mathbf{Samp}(\psi_\sigma^n, seed)$
- 6: $e_2 \leftarrow \mathbf{Samp}(\psi_\sigma^{l_v}, seed)$
- 7: $c_1 \leftarrow ar + e_1 \in R_q$
- 8: $c_2 \leftarrow (br)_{l_v} + e_2 + \lfloor \frac{q}{2} \rfloor \hat{m} \in \mathbb{Z}_q^{l_v}$
- 9: **if** LAC-256
- 10: $c_2 \leftarrow c_2 || c_2$ //D2 encoding
- 11: **end if**
- 12: $c_2 \leftarrow \mathbf{Compress}(c_2)$
- 13: **return** $c = (c_1, c_2)$

Algorithm 3.*CPA.Decrypt* $(s_k, c = (c_1, c_2))$

Ensure: Plaintext m

- 1: $c_2 \leftarrow \mathbf{Decompress}(c_2)$
- 2: $\hat{M} \leftarrow c_2 - (c_1 s_k)_{l_v} \in \mathbb{Z}_q^{l_v}$
- 3: **for** $i = 0$ to $l_v - 1$ **do**
- 4: **if** $\frac{q}{4} \leq \hat{M}_i < \frac{3q}{4}$ **then**
- 5: $\hat{m}_i \leftarrow 1$
- 6: **else**
- 7: $\hat{m}_i \leftarrow 0$
- 8: **end if**
- 9: **end for**
- 10: $m \leftarrow \mathbf{BCHDecode}(\hat{m})$
- 11: **return** m

Algorithm 4.*CPA.Decrypt256* $(s_k, c = (c_1, c_2))$

Ensure: Plaintext m

- 1: $c_2 \leftarrow \mathbf{Decompress}(c_2)$
- 2: $\hat{M} \leftarrow c_2 - (c_1 s_k)_{2l_v} \in \mathbb{Z}_q^{2l_v}$
- 3: **for** $i = 0$ to $l_v - 1$ **do** //D2 Decoding
- 4: $tmp_1, tmp_2 := \hat{M}[i], \hat{M}[i + l_v]$
- 5: **if** $tmp_1 < \frac{q}{2}$
- 6: $tmp_1 \leftarrow q - tmp_1$
- 7: **else if** $tmp_2 < \frac{q}{2}$
- 8: $tmp_2 \leftarrow q - tmp_2$
- 9: **end if**
- 10: **if** $tmp_1 + tmp_2 - q < \frac{q}{2}$
- 11: $\hat{m}_i \leftarrow 1$
- 12: **else**
- 13: $\hat{m}_i \leftarrow 0$
- 14: **end if**
- 15: **end for**
- 16: $m \leftarrow \mathbf{BCHDecode}(\hat{m})$
- 17: **return** m

Subroutines

BCHEncode and BCHDecode. The function **BCHEncode** takes as input a message m of length l_m , pads it with $(k - l_m)$ zeros, where k is the dimension of the BCH code, and returns the corresponding value c on the code. The function **BCHDecode** takes as input a message \hat{c} of length $n - 1$, retrieves the codeword c closest to \hat{c} and returns m such that $c = mG$, where G is the generator matrix of the code.

Compress and Decompress. The function **Compress** takes as input a variable $c = (c_0, \dots, c_{len_c})$ where each coefficient c_i is a 8-bits number and returns $c' = (c'_0, \dots, c'_{len_c})$ where each c'_i is a 4 bits number obtained by keeping the highest 4 bits of c_i .

The function **Decompress** takes as input a variable $c' = (c'_0, \dots, c'_{len_c})$ where each coefficient c'_i is a 4-bit number, and returns $\tilde{c} = (\tilde{c}_0, \dots, \tilde{c}_{len_c})$ where each \tilde{c}_i is a 8 bits number obtained by padding each coefficient c'_i with 4 zero bits.

Parameters

In the following we denote the secret key sk by s . Recall that LAC is a RLWE public-key encryption scheme on $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, with input messages of length l_m .

LAC uses different parameters for its three algorithms:

Name	n	q	Distrib	l_m	l_v	Code(BCH) $[n', k, d]$	D2
LAC-128	512	251	ψ_1	256	$l_m + 144$	[511, 367, 33]	No
LAC-192	1024	251	$\psi_{\frac{1}{2}}$	256	$l_m + 72$	[511, 439, 17]	No
LAC-256	1024	251	ψ_1	256	$l_m + 144$	[511, 367, 33]	Yes

The value l_v depends on the BCH code. Let G be a generator matrix of the BCH code C . By the construction of LAC, G is on systematic form $G = (Id_k | A_{n'-k})$. In fact, we cannot keep only l_v bits of a codeword without this condition. The **BCHEncode** function takes as input a message m of length l_m and pads it with $(k - l_m)$ zeros. We obtain

$$(m_1, \dots, m_{l_m}, 0_1, \dots, 0_{k-l_m})G = (m_1, \dots, m_{l_m}, 0_1, \dots, 0_{k-l_m} | mA_{n'-k}) = c$$

We omit the $(k - l_m)$ zeros of c then $l_v = l_m + (n' - k)$.

LAC Key Exchange

We describe the LAC Key Exchange introduced in [19], based on the CPA version of the LAC public-key encryption scheme.

Alice	Bob
$(pk, s) \leftarrow \mathbf{CPA.KeyGen}()$	
	\xrightarrow{pk}
	$r \leftarrow U(\{0, 1\}^{l_m})$
	$c \leftarrow \mathbf{CPA.Encrypt}(pk, r)$
	$Key_B \leftarrow H(pk, r) \in \{0, 1\}^{l_k}$
	\xleftarrow{c}
$r' \leftarrow \mathbf{CPA.Decrypt}(s, c)$	
$Key_A \leftarrow H(pk, r') \in \{0, 1\}^{l_k}$	

If Key Exchange succeeds then $r' = r$ and $Key_B = Key_A$.

3 Attack on LAC Key Exchange

In this section, we present the main result of this paper. We start by defining the scenario of the attack by introducing the oracle defined in [4].

3.1 Attack Model

Suppose that Alice does a misuse of the Key Exchange Mechanism by caching her secret s . More precisely:

Assumption 1. *Alice keeps her secret key constant for several CPA key establishments requests.*

Eve is a malicious active adversary who acts as Bob and can cheat and generate c that is not the encryption of a random r . To mount the active attack, we suppose that Eve has access to a session key mismatch oracle defined as follow.

Definition 1. *A key mismatch oracle outputs a bit of information on the possible mismatch at the end of the key encapsulation mechanism. In the LAC context, this oracle, denoted \mathcal{O} , takes any message c and any session key guess μ as input and outputs:*

$$\mathcal{O}(c, \mu) = \begin{cases} 1 & \text{if } H(pk, CPA.Decrypt(s, c)) = \mu \\ -1 & \text{otherwise} \end{cases}$$

This oracle can also be used by Bob during an honest key exchange with Alice, when he verifies the match between his session key and Alice's one.

The idea of the attack mounted by Eve is to send forged ciphertexts to Alice to ensure that she obtains information on some coefficients of Alice's secret key. As Eve knows that $c = (c_1, c_2)$ and s are used during the decryption algorithms (s is multiplied by c_1), she will mount an attack using this fact and following four mains steps:

- Choose a session key μ .
- Construct c_1 such that some coefficients of the secret key are exposed.
- Construct c_2 depending of μ such that the result of Alice's decryption can be monitored as a function of the key guess.
- Call to the oracle \mathcal{O} to obtain information about our key guesses.

The following section shows how to choose appropriate (c, μ) to retrieve information on s . We assume that Eve has access to the oracle \mathcal{O} .

3.2 Attack on LAC-128-KE and LAC-192-KE

First, we use a simplified version where we do not consider *Compress* and *Decompress* functions. We follow the different steps of the decryption Algorithm 3.

Simplified Version

In this first result, we show how one can forge a LAC ciphertext in order to impose which plaintext will be obtained after decryption. To do so, we need to forge c such that the impact of the secret key during the decryption is under our control.

Proposition 1. *Assume that Eve forges $c = (c_1, c_2)$ such that :*

- $c_1 = -ax^{n-w}$ where w is an integer $0 \leq w < n$ and $0 \leq a < \frac{q}{4}$
- $c_2 = (\alpha_0, \dots, \alpha_{l_v-1})$ where $\alpha_i = \frac{q}{2}$ or 0 for all i in $[0, l_v - 1]$.

Then she can determine the plaintext m that Alice will obtain after decryption.

Proof. When Alice deciphers Eve’s ciphertext she:

1. Computes $\widehat{M} = c_2 - (c_1s)_{l_v}$
2. Compares each coefficient of \widehat{M} to $\frac{q}{4}$ and $\frac{3q}{4}$ to define \widehat{m}
3. Retrieves m using **BCHDecode** algorithm on \widehat{m}

Let $c_1 = -ax^{n-w}$ and $s = s_0 + s_1x^1 + \dots + s_{n-1}x^{n-1}$ then

$$c_1s = as_w + as_{w+1}x + \dots + as_{n-1}x^{n-w-1} - as_0x^{n-w} - \dots - as_{w-1}x^{n-1}$$

and the polynomial c_1s can be represented as the vector $(as_w, \dots, -as_{w-1})$

During the computation of \widehat{M} , two cases are possible:

- $w < l_v$ then $\widehat{M} = c_2 - (c_1s)_{l_v} = (\alpha_0 - as_w, \dots, \alpha_w + as_0, \dots, \alpha_{l_v-1} + as_{w+l_v-1})$
- $w \geq l_v$ then $\widehat{M} = c_2 - (c_1s)_{l_v} = (\alpha_0 - as_w, \alpha_1 - as_{w+1}, \dots, \alpha_{l_v-1} - as_{w+l_v-1})$

After this computation each coefficient of \widehat{M} is compared to $\frac{q}{4} \leq \widehat{M}_i < \frac{3q}{4}$. Recall that since $s \leftarrow \psi_\sigma^n$, each of its coefficients belongs to $\{-1, 0, 1\}$. Let i be an integer such that $0 \leq i < n$ and $j \equiv n - w + i \pmod n$.

If $\alpha_i = \frac{q}{2}$ one gets:

$$\alpha_i \mp as_j = \begin{cases} \frac{q \mp 2a}{2} & \text{if } s_j = \pm 1 \\ \frac{q}{2} & \text{if } s_j = 0 \\ \frac{q \pm 2a}{2} & \text{if } s_j = \mp 1 \end{cases}$$

If $\alpha_i = 0$ one gets:

$$\alpha_i \mp as_j = \begin{cases} \pm a & \text{if } s_j = \mp 1 \\ 0 & \text{if } s_j = 0 \\ \mp a & \text{if } s_j = \pm 1 \end{cases}$$

In the first case, the three possible values for $\alpha_i \mp as_j$ lie in $[\frac{q}{4}, \frac{3q}{4}[$ if $0 \leq a \leq \frac{q}{4}$. In the case $\alpha_i = 0$, the three possible values do not lie in $[\frac{q}{4}, \frac{3q}{4}[$ when $0 \leq a < \frac{q}{4}$ or $\frac{3q}{4} \leq a \leq q$.

Thus, Eve can choose $a < \frac{q}{4}$ and $\alpha_i = \frac{q}{2}$ or 0 to determine what Alice will obtain on the first l_v coordinates of \widehat{m} . Then, Eve can deduce, by applying BCH decoding, what Alice obtains at the end of the decryption procedure.

The next example explains how one can use Proposition 1.

Example 1. Suppose that Eve wants that Alice will obtain, after decryption, the message $m = \mathbf{BCHDecode}(1, 0, 1, 1, 0, \dots, 0)$. Then she forges $c = (c_1, c_2)$ such that:

- $c_1 = -\frac{q}{5}x^n = \frac{q}{5}$ on R_q . In fact Eve can take any c_1 such that $c_1 = -ax^{n-w}$ with $0 \leq a < \frac{q}{4}$
- $c_2 = (\frac{q}{2}, 0, \frac{q}{2}, \frac{q}{2}, 0, \dots, 0)$

From c Alice first computes:

$$\begin{aligned} \widehat{M} &= c_2 - (c_1s)_{l_v} \\ &= \left(\frac{q}{2}, 0, \frac{q}{2}, \frac{q}{2}, 0, \dots, 0\right) - \frac{q}{5}(s_0, s_1, \dots, s_{l_v}), \text{ } s_i \text{ belongs to } \{-1, 0, 1\} \\ &= \left(\frac{q}{2} - \frac{q}{5}s_0, -\frac{q}{5}s_1, \frac{q}{2} - \frac{q}{5}s_2, \frac{q}{2} - \frac{q}{5}s_3, -\frac{q}{5}s_4, \dots, -\frac{q}{5}s_{l_v}\right) \end{aligned}$$

Then, Alice compares each coefficients of \widehat{M} to $\frac{q}{4}$ and $\frac{3q}{4}$. She obtains (see proof of Proposition 1):

$$\widehat{m} = (1, 0, 1, 1, 0, \dots, 0)$$

At the end, Alice obtains m by applying $\mathbf{BCHDecode}$ algorithm to \widehat{m} . Thus, Eve had forged c such that Alice has $m = \mathbf{BCHDecode}(1, 0, 1, 1, 0, \dots, 0)$.

With Proposition 1 we construct a ciphertext such that the secret key has no impact during decryption. Now Eve needs to construct forged ciphertexts that allow a key guessing strategy in order to retrieve the secret key. Thus, we need that the secret key has an impact during decryption if and only if we did a good key guess.

Proposition 2. *Let s'_w be a guess done by Eve on the w -th coefficient of the secret key s , where $0 \leq w < n$. Assume $s_w = 1$ or -1 . If Eve forges $c = (c_1, c_2)$ as given in Proposition 1 and modify the first coordinate of c_2 such that:*

- $c_2 = (as'_w, \alpha_1, \dots, \alpha_{l_v-1})$ with $\frac{q}{8} < a < \frac{q}{4}$.

Then she can verify her key guess from the plaintext computed by Alice from c .

Proof. Suppose that Eve wants to retrieve the w -th coefficient of s . When Alice will decipher Eve ciphertext she first computes:

$$\widehat{M} = c_2 - (c_1s)_{l_v} = (as'_w - as_w, \alpha_1 - as_{w+1}, \dots)$$

According to Proposition 1, Eve can determine what Alice will obtain for every coefficient different from her guess s'_w . Let see what happens with this coefficient by analysing $as'_w - as_w$.

$$as'_w - as_w = \begin{cases} 0 & \text{if } s'_w = s_w \\ 2a & \text{if } s'_w = 1 \text{ and } s_w = -1 \\ -2a & \text{if } s'_w = -1 \text{ and } s_w = 1 \\ \mp a & \text{if } s'_w = 0 \text{ or } s_w = 0 \end{cases}$$

Let $\frac{q}{8} < a < \frac{q}{4}$ then $\frac{q}{4} < 2a < \frac{q}{2}$ and $-2a = q - 2a$ satisfies $\frac{q}{2} < q - 2a < \frac{3q}{4}$. Then with $a \in]\frac{q}{8}, \frac{q}{4}[$. The key guess is good (resp. wrong) when a 1 (resp. 0) is returned at the first coordinate of \widehat{m} . Hence Eve can effectively determines what Alice obtained by applying **BCHDecode** algorithm to \widehat{m} and thus deterministically verifies her key guess from \widehat{m} .

Proposition 2 ensures that if Eve guessed the good key then Alice will obtains $m = \mathbf{BCHDecode}(1, \dots)$. Otherwise, she will obtain $m = \mathbf{BCHDecode}(0, \dots)$. Computational details are given in the following example.

Example 2. Suppose that Eve wants to learn information about the first bit of Alice’s secret key. Eve forges $c = (c_1, c_2)$ such that:

- $c_1 = -\frac{q}{5}x^n = \frac{q}{5}$ on R_q .
- $c_2 = (\frac{q}{5}s'_0, 0, \frac{q}{2}, \frac{q}{2}, 0, \dots, 0)$ where s'_w is Eve’s key guess.

As in Example 1, Alice first computes $\widehat{M} = c_2 - (c_1s)_{l_v} = (\frac{q}{5}s'_0 - \frac{q}{5}s_0, -\frac{q}{5}s_1, \frac{q}{2} - \frac{q}{5}s_2, \frac{q}{2} - \frac{q}{5}s_3, -\frac{q}{5}s_4, \dots, -\frac{q}{5}s_{l_v})$ where s_i belongs to $\{-1, 0, 1\}$. Then, Alice compares each coefficients of \widehat{M} to $\frac{q}{4}$ and $\frac{3q}{4}$. She obtains (see proof of Proposition 2):

$$\begin{aligned} \widehat{m} &= (1, 0, 1, 1, 0, \dots, 0) \text{ if } s'_0 = -s_0 \text{ and } s_0 \neq 0 \\ \widehat{m} &= (0, 0, 1, 1, 0, \dots, 0) \text{ otherwise} \end{aligned}$$

At the end, Alice obtains m by applying **BCHDecode** algorithm to \widehat{m} . Thus, Eve did the good key guess if Alice gets $m = \mathbf{BCHDecode}(1, 0, 1, 1, 0, \dots, 0)$.

Proposition 2 already gives interesting information to Eve but it is not enough to mount an attack since Eve needs a way to verify if Alice obtains:

- either $m = \mathbf{BCHDecode}(1, 0, 1, 1, 0, \dots, 0)$
- or $m = \mathbf{BCHDecode}(0, 0, 1, 1, 0, \dots, 0)$

without knowing m . Moreover, most of the time **BCHDecode**(1, 0, 1, 1, 0, ..., 0) will not differ from **BCHDecode**(0, 0, 1, 1, 0, ..., 0).

To overcome these issues we need to instantiate precisely the oracle given in Definition 1 using Proposition 2.

Instantiation of the Oracle. The oracle defined in Definition 1 gives information about the success of a key session establishment between Alice and Bob. Eve can use such an oracle with the help of Proposition 2 and the BCH code decryption failure to overcome issues mentioned above.

In the sequel, we show how Eve can practically mount an attack by forging specific inputs to this oracle and deduce information on Alice’s secret key. The following theorem and its proof detail this construction then Algorithm 5 and Algorithm 6 formally describe the attack.

Theorem 1. *Let $s'_w \in \{-1, 1\}$ be the guessed value of s_w ($0 \leq w < n$) done by Eve. If Eve takes a session key $\mu_{s'_w}$ then she can forge $c_{s'_w} = (c_1, c_2)$ depending of $\mu_{s'_w}$ by using properties given in Proposition 2 such that by calling $\mathcal{O}(c_{s'_w}, \mu_{s'_w})$ with $s'_w \in \{-1, 1\}$, she retrieves the w -th coefficient of s . In consequence, Eve needs at most 2 calls to the oracle in order to retrieve a coefficient of Alice’s secret key.*

Proof. According to Proposition 2, Eve can monitor Alice’s decryption procedure if she does the good key guess.

An error-correction code can correct at most $\frac{d-1}{2}$ errors (where d is the minimal Hamming distance of the BCH code). The idea is that after comparison with $\frac{q}{4}$ and $\frac{3q}{4}$, \hat{m} is a codeword with $\frac{d}{2}$ errors if Eve did the wrong key guess, causing a decoding error. Suppose Eve wants to retrieve the w -th coefficient of s :

1. Eve chooses a codeword called *cdword* with a 1 at the first coordinate such that $cdword = mG$ where G is the generator matrix of the BCH code
2. Eve injects $\frac{d-1}{2}$ errors to *cdword* at any coordinate except the first one
3. Eve chooses a verifying $\frac{q}{8} < a < \frac{q}{4}$ according to Proposition 2
4. Eve constructs c_1, c_2 with her key guess at the first bit of c_2 : $c_2[0] = as'_w$ and such that after comparison with $\frac{q}{4}$ and $\frac{3q}{4}$, Alice retrieves *cdword* with $\frac{d-1}{2}$ errors or *cdword* with $\frac{d}{2}$ errors
5. Eve sends $c = (c_1, c_2)$ to Alice

With this construction, Alice obtains a codeword with $\frac{d}{2}$ errors if Eve provides a wrong key guess. At this point, Eve’s session key is $sess_E = H(pk, m)$ and Alice’s session key $sess_A$ depends on Eve’s key guess. Eve can verify whether she did the correct key guess with the oracle as follow:

If $s'_w = 1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = -1$ and $sess_A = sess_E$
Else If $s'_w = -1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = 1$ and $sess_A = sess_E$
Otherwise $s_w = 0$

Algorithm 5 and Algorithm 6 are based on the construction described in the proof of Theorem 1. Here, we fix the constant a to $\frac{q}{7}$ in the construction of c_1 .

Algorithm 5. `forge(hyp,bit)`

Ensure: Forge ciphertext $c = (c_1, c_2)$

- 1: $c_1 := -\frac{q}{7}x^{n-bit}$
- 2: $m := [0 : \text{for } i := 0 \text{ to } 255]$
- 3: $m[0] := 1$
- 4: $codeword := (m||0..0)G$
- 5: Add $\frac{d-1}{2}$ errors to codeword (but not on codeword[0])
- 6: **For** $i = 0$ **to** $\text{Len}(codeword)$:
- 7: **if** $i == 0$:
- 8: $c_2[0] \leftarrow \text{hyp} \times \frac{q}{7}$
- 9: **else if** $codeword[i] == 1$:
- 10: $c_2[i] \leftarrow \frac{q}{2}$
- 11: **else**
- 12: $c_2[i] \leftarrow 0$
- 13: **end if**
- 14: **end for**
- 15: **Return** $(m, c = (c_1, c_2))$

Algorithm 6. `recoverOneBit(bit)`

Ensure: A bit of s

- 1: $m, c := \text{forge}(-1, bit)$
- 2: **If** $\mathcal{O}(c, m) == 1$:
- 3: **Return** 1
- 4: **end if**
- 5: $m, c := \text{forge}(1, bit)$
- 6: **If** $\mathcal{O}(c, m) == 1$:
- 7: **Return** -1
- 8: **end if**
- 9: **Return** 0

Using Theorem 1, a key of length n can be fully recovered with at most $2 \times n$ requests to the oracle. LAC-128 works with keys of length $n = 512$ and LAC-192 with length $n = 1024$.

Full Version

The subroutine *Compress* removes the 4 lower bits of each coeff of c_2 . They are replaced by 4 zero-bit when the subroutine *Decompress* is applied at the beginning of the decryption process. Thus, each coefficient of c_2 can be only equal to 16, 32, 64, 128 and any sum of these values.

For c_2 in our attack, we only consider the values $\frac{q}{7}$, $-\frac{q}{7}$ and $\frac{q}{2}$. In our implementation [14] we approximate $\frac{q}{7} \approx 32$, $-\frac{q}{7} \approx 128 + 64 + 16 = 210$ and $\frac{q}{2} \approx 128$. Proposition 2 is still verified and we still retrieve s with at most $2 \times n$ requests to the oracle by the Theorem 1.

In comparison of the recent work of Huguenin-Dumittan *et al.* in [10], our upper-bound for LAC128 is 2 times less than theirs. Indeed, they need at most 2^{11} queries to retrieve the entire secret key, while we need at most 2^{10} queries.

Implementation Results

We have developed a C implementation of the attack (see [14]). To assess its efficiency we use the reference code of LAC [19] as a target. In the following, we present practical results on the average of 1000 attacks launched on 1000 random secret keys for LAC-128 and LAC-192. Timing results have been evaluated on core i5-8350U at 1.90 GHz.

The size of LAC-192 secret key is 2 times larger than LAC-128 one, but the number of required request to retrieve sk is more than 2 times larger. This is due to a different probability distribution between these two levels of security.

	Nb of coeff of sk	Average oracle requests	Average time
LAC-128	512	896	2, 94 ms
LAC-192	1024	1920	15, 53 ms

In average we need $1,75 \times 512$ oracle requests for LAC-128 and $1,875 \times 1024$ requests for LAC-192. For both cases, the practical result is less than the upper bound of $2 \times n$ where $n = 512$ or 1024 .

4 Attack on LAC-256-KE

4.1 Attack on LAC-256-KE

Since LAC-256 encryption uses $D2$ encoding, the decryption procedure is slightly different. Let $c = (c_1, c_2)$, $D2$ encoding duplicates the coordinate of c_2 : $c_2 = (c_2 || c_2)$. The use of this encoding allows to decrease decoding errors.

In Attack on LAC-128/192 we forged c_1 as a monomial to avoid linear combination between coefficients of s during computation of c_1s . This allows to do key guess on only one coefficient of s . But, despite the use of a monomial for c_1 , $D2$ encoding ensures that each coefficient of c_1s is a linear combination of at least 2 coefficients of s . It implies that we need to do key guesses on two coefficients of s . In this section, we adapt our previous attack to allow to do two key guesses rather than one. The attack procedure is the same as previously:

- Choose a session key μ .
- Construct c_1 such that some coefficients of the secret key are exposed.
- Construct c_2 depending of μ such that the result of Alice’s decryption can be monitored as a function of the key guess.
- Call to the oracle \mathcal{O} to obtain information about our key guesses.

For the sake of clarity all proposition proofs are in Appendix.

CPA.Decrypt256 Description

The first step of the decryption it’s to compute $\widehat{M} = c_2 - (c_1s)_{2l_v}$ as previously. However the comparison is different for LAC-256. The decryption algorithm considers two cases

Case 1. If $\widehat{M}[i]$ and $\widehat{M}[i + l_v] < \frac{q}{2}$ or $\widehat{M}[i]$ and $\widehat{M}[i + l_v] \geq \frac{q}{2}$ then algorithm

CPA.Decrypt256 checks whether: $\frac{\widehat{M}[i] + \widehat{M}[i + l_v]}{2} \in]\frac{q}{4}, \frac{3q}{4}[$

Case 2. If $\widehat{M}[i] < \frac{q}{2}$ and $\widehat{M}[i + l_v] \geq \frac{q}{2}$ or $\widehat{M}[i] \geq \frac{q}{2}$ and $\widehat{M}[i + l_v] < \frac{q}{2}$ then

CPA.Decrypt256 checks whether $\frac{|\widehat{M}[i] - \widehat{M}[i + l_v]|}{2} \in]0, \frac{q}{4}[$

In the following we notice when we are in the case 1 or 2.

4.2 Attack on LAC-256-KE Simplified

As previously we first use a simplified version where we do not consider *Compress* and *Decompress* subroutines.

Proposition 3. *Assume that Eve forges $c = (c_1, c_2)$ such that:*

- $c_1 = -ax^{n-w}$ where w is an integer $0 \leq w < (n - l_v)$ and $0 \leq a < \frac{q}{4}$
- $c_2 = (\alpha_0, \dots, \alpha_{l_v-1}, \alpha_{l_v}, \dots, \alpha_{2l_v-1})$ where $\alpha_i = \frac{q}{2}$ or 0 for all i in $[0, 2l_v - 1]$

Then she can determine the plaintext m that Alice obtains after decryption.

Example 3. Suppose that Eve wants that Alice obtains, after decryption, the plaintext $m = \mathbf{BCHDecode}(1, 1, 0, 1, 0, \dots, 0)$. Eve forges $c = (c_1, c_2)$ such that:

- $c_1 = -\frac{q}{5}x^n = \frac{q}{5}$ on R_q .
- $c_2 = (\frac{q}{2}, \frac{q}{2}, 0, \frac{q}{2}, 0, \dots, 0 \parallel \frac{q}{2}, \frac{q}{2}, 0, \frac{q}{2}, 0, \dots, 0)$. The symbol \parallel delimit the l_v first part to the l_v second part of c_2 (we duplicate c_2 due to $D2$ encoding in Algorithm 2). The two parts are symmetric .

When Alice decipheres c , she computes $\widehat{M} = c_2 - (c_1 s)_{2l_v}$ and uses the comparison procedure describes in Algorithm 4 to obtain \widehat{m} of length l_v . If c_1 and c_2 are constructed according to Proposition 3, then (cf Proof 5):

- If $c_2[i] = c_2[i + l_v] = \frac{q}{2}$ then $\widehat{m}[i] = 1$
- If $c_2[i] = c_2[i + l_v] = 0$ then $\widehat{m}[i] = 0$

Then, with our $c_2 = (\frac{q}{2}, \frac{q}{2}, 0, \frac{q}{2}, 0, \dots, 0 \parallel \frac{q}{2}, \frac{q}{2}, 0, \frac{q}{2}, 0, \dots, 0)$ Alice obtains $\widehat{m} = (1, 1, 0, 1, 0, \dots, 0)$. Thus, Alice retrieves $m = \mathbf{BCHDecode}(1, 1, 0, 1, 0, \dots, 0)$.

So Eve can choose $a < \frac{q}{4}$ and $\alpha_i = \frac{q}{2}$ or 0 to know what Alice obtains on the l_v coordinates of \widehat{m} and then Eve can deduce what Alice obtains at the end of decryption for m . Eve needs to construct forged ciphertexts which allow to verify her key guesses.

Proposition 4. *Let s'_w and s'_{w+l_v} be guesses done by Eve on the w -th and $w+l_v$ coefficients of the secret key s . Assume $s_w, s_{w+l_v} = 1$ or -1 . If Eve forges $c = (c_1, c_2)$ as given in Proposition 3 and modify the first and l_v -th coordinates of c_2 such that:*

- $c_2 = (as'_w, \alpha_1, \dots, \alpha_{l_v-1}, as'_{w+l_v}, \dots, \alpha_{l_v-1})$ with $\frac{q}{8} < a < \frac{q}{4}$.

Then she can verify her key guesses from the plaintext computed by Alice from c .

Example 4. Suppose that Eve wants to learn information about the first and the l_v -th bit of Alice's secret key. Eve forges $c = (c_1, c_2)$ such that:

- $c_1 = -\frac{q}{5}x^n = \frac{q}{5}$ on R_q .
- $c_2 = (\frac{q}{5}s'_0, \frac{q}{2}, 0, \frac{q}{2}, 0, \dots, 0 \parallel \frac{q}{5}s'_{l_v}, \frac{q}{2}, 0, \frac{q}{2}, 0, \dots, 0)$ where s'_0 and s'_{l_v} are key guesses

When Alice deciphers c she computes $\widehat{M} = c_2 - (c_1 s)_{2l_v}$ and uses the comparison procedure describes in Algorithm 4 to obtain \widehat{m} of length l_v . If c_1, c_2, s'_0 and s'_{l_v} are constructed according to Proposition 4, then (cf Proof 5):

- If $s'_0 = -s_0$ and $s'_{l_v} = -s_{l_v}$ then $\widehat{m}[0] = 1$
- Else $\widehat{m}[0] = 0$
- The value of the others coefficients of \widehat{m} are determined as in the previous example

If Eve does correct key guesses then Alice obtains $\widehat{m} = (1, 1, 0, 1, 0, \dots, 0)$. Otherwise, Alice obtains $\widehat{m} = (0, 1, 0, 1, 0, \dots, 0)$

Proposition 4 ensures that Eve can know what Alice obtains if Alice’s secrets coefficients are different from 0. Let see what happens when one of the two coefficient is equal to 0.

Proposition 5. *Let s'_w and s'_{w+l_v} be guesses done by Eve on the w -th and $w+l_v$ coefficients of the secret key s . Assume $s_w = 0$ or $s_{w+l_v} = 0$. If Eve forges $c = (c_1, c_2)$ as given in Proposition 3 and modify the first and l_v -th coordinates of c_2 such that:*

- $c_2 = (as'_w, \alpha_1, \dots, \alpha_{l_v-1}, as'_{w+l_v}, \dots, \alpha_{l_v-1})$ with $\frac{q}{6} < a < \frac{q}{4}$.

Then she can verify her key guesses from the plaintext computed by Alice from c .

Proposition 5 works like Proposition 4 but for the case where one of the two targeted coefficient is equal to 0. However, as previously, Proposition 4 and Proposition 5 are not enough to mount an attack for the same reasons:

- Eve needs a way to verify what Alice obtains.
- A bit of difference on \widehat{m} is corrected by the BCH code. Thus, at the end of the decryption procedure Alice and Eve have the same plaintext.

Nonetheless, Eve can use Proposition 4 and Proposition 5, the BCH code decryption failure and the oracle to overcome these issues.

Theorem 2. *Let $s'_w, s'_{w+l_v} \in \{-1, 1\}$ be the guessed values of s_w and s_{w+l_v} done by Eve. If Eve takes a session key $\mu_{s'_w, s'_{w+l_v}}$ then she can forge $c_{s'_w, s'_{w+l_v}} = (c_1, c_2)$ depending of $\mu_{s'_w, s'_{w+l_v}}$ by using properties given in Proposition 2 such that by calling $\mathcal{O}(c_{s'_w, s'_{w+l_v}}, \mu_{s'_w, s'_{w+l_v}})$ with $s'_w, s'_{w+l_v} \in \{-1, 0, 1\}$, she retrieves the w -th and $w+l_v$ -th coefficients of s . In consequence, Eve needs at most $8 \times (n - l_v)$ calls to the oracle in order to retrieve two coefficients of Alice’s secret key.*

Proof. The idea is the same as LAC-128 and 192, Eve takes c_2 to ensure, after comparison in CPA.Decrypt256, that \widehat{m} is a codeword with $\frac{d}{2}$ errors if she did a wrong key guess. Since at most $\frac{d-1}{2}$ errors can be corrected, a decoding errors occurs.

According to Proposition 4 and Proposition 5, Eve can monitor Alice’s decryption procedure if she does the good key guess.

Suppose Eve wants to retrieve the w -th and the $(w+l_v)$ -th coefficients of s :

1. Eve chooses a codeword called *cdword* with a 1 at the first coordinate such that $cdword = mG$ where G is the generator matrix of the BCH code
2. Eve injects $\frac{d-1}{2}$ errors to *cdword* at any coordinate except the first one
3. Eve chooses a verifying $\frac{q}{8} < a < \frac{q}{4}$ if she is on the case of Proposition 4 or $\frac{q}{6} < a < \frac{q}{4}$ if she is on the case of Proposition 5
4. Eve constructs c_1 and c_2 with her key guesses at the first and l_v -th coefficient of c_2 : $c_2[0] = as'_w$ and $c_2[l_v] = as'_{w+l_v}$ and such that after comparison, Alice retrieves *cdword* with $\frac{d-1}{2}$ errors or *cdword* with d errors
5. Eve sends $c = (c_1, c_2)$ to Alice

With this construction Alice obtains a codeword with $\frac{d}{2}$ errors if Eve does a wrong key guess. At this point, Eve's session key is $sess_E = H(pk, m)$ and Alice's session key $sess_A$ depends on Eve's key guesses. Eve can verify if she did a good key guess with the oracle.

First Eve determines if s_w and s_{w+l_v} are different from 0 (see Proposition 4):

- If** $s'_w = 1, s'_{w+l_v} = 1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = -1$ and $s_{w+l_v} = -1$
- Else If** $s'_w = -1, s'_{w+l_v} = -1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = 1$ and $s_{w+l_v} = 1$
- Else If** $s'_w = 1, s'_{w+l_v} = -1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = -1$ and $s_{w+l_v} = 1$
- Else If** $s'_w = -1, s'_{w+l_v} = 1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = 1$ and $s_{w+l_v} = -1$

If the oracle does not return 1, then Eve determines which coefficient is equal to 0 (see Proposition 5):

- If** $s'_w = 1, s'_{w+l_v} = 1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = -1$ and $s_{w+l_v} = 0$
- or $s_w = 0$ and $s_{w+l_v} = -1$
- If** $s'_w = -1, s'_{w+l_v} = 1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = 0$ and $s_{w+l_v} = -1$
- Else If** $\mathcal{O}(c, sess_E) = -1$ then $s_w = -1$ and $s_{w+l_v} = 0$
- Else If** $s'_w = -1, s'_{w+l_v} = -1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = 1$ and $s_{w+l_v} = 0$
- or $s_w = 0$ and $s_{w+l_v} = 1$
- If** $s'_w = 1, s'_{w+l_v} = 1$ and $\mathcal{O}(c, sess_E) = 1$ then $s_w = 0$ and $s_{w+l_v} = 1$
- Else if** $\mathcal{O}(c, sess_E) = -1$ then $s_w = 1$ and $s_{w+l_v} = 0$
- Otherwise** $s_w = 0$ and $s_{w+l_v} = 0$

Eve can apply this procedure for $0 \leq w < (n - l_v)$ to retrieve the entire secret key.

To recover the entire key we need at most $8 \times (n - l_v)$ requests to the oracle due to Theorem 2, where $l_v = 400$ and $n = 1024$.

Full Version

The subroutine **Compress** removes the 4 lowest bits of each coefficient of c_2 . They are replaced by 4 zero-bits when the subroutine **Decompress** is applied at the

beginning of the decryption process. So each coefficient of c_2 can be only equal to 16, 32, 64, 128 and any sum of these values.

For c_2 in our attack we choose $a \approx \frac{q}{7}$ for Proposition 4 and $a \approx \frac{q}{5}$ for Proposition 5. Then, we only consider the values $\frac{q}{7}$, $-\frac{q}{7}$, $\frac{q}{5}$, $-\frac{q}{5}$ and $\frac{q}{2}$. In our implementation [14] we approximate $\frac{q}{7} \approx 32$, $-\frac{q}{7} \approx 128 + 64 + 16 = 210$ or $-\frac{q}{7} \approx 128 + 64 + 32 = 224$ (we use two different values to compensate the approximation), $\frac{q}{5} \approx 16 + 32 = 48$, $-\frac{q}{5} \approx 128 + 64 = 192$ and $\frac{q}{2} \approx 128$. Proposition 4 and Proposition 5 are still verified and we still retrieve s with at most $8 \times (n - l_v)$ requests to the oracle by the Theorem 2.

Implementation Results

We assess our attack implementation [14] plug in the reference code of LAC. Following results are the average of 1000 attacks launched on 1000 random secret keys for LAC-256. Timing results have been evaluated on core i5-8350U at 1.90 GHz.

	Nb of coeff of sk	Average oracle requests	Average time
LAC-256	1024	3355	30, 31 ms

In average we need $5, 4 \times (1024 - 400)$ oracle requests that is much less than the upper bound of $8 \times (n - l_v)$ requests

5 Conclusion

In this paper, we show how to mount an attack on CPA version of LAC-KE when the same secret key is reused. Moreover, on constrained environment this attack can be applied on the CCA version by applying physical attack on the Fujisaki-Okamoto transformation as shown in [7, 16]. We prove that this attack needs at most 8×1024 queries of key exchanges. This low number of queries to recover the secret confirmed the necessity to not reuse the same private key even for a very small number of key exchanges. One can compare this number with the key mismatch attack on NewHope in [11] that requires 882, 794 queries and the one on Kyber in [17] that requires $2, 4 \times 1024$ queries. Hence, in the context of key reuse, LAC-256 is much less resilient than NewHope but a little more resilient than Kyber. It is important to note that this situation is a misuse and thus, LAC is still believed to be safe when a fresh secret key is used for each exchange. (The same remark applies to NewHope and Kyber.)

Appendix

Proof of Proposition 3

Proof. Assuming Alice receives $c = (c_1, c_2)$ then she:

1. Computes $\widehat{M} = c_2 - (c_1 s)_{2l_v}$
2. Compares $\frac{q}{4} < \frac{\widehat{M}[i] + \widehat{M}[i+l_v]}{2} < \frac{3q}{4}$ or $0 < \frac{|\widehat{M}[i] - \widehat{M}[i+l_v]|}{2} < \frac{q}{4}$ for $i = 0$ to $l_v - 1$ to define each coefficient of \widehat{m}
3. Retrieves m using **BCHDecode** algorithm on \widehat{m}

Let $c_1 = -ax^{n-w}$ and $s = s_0 + s_1x^1 + \dots + s_{n-1}x^{n-1}$ then $c_1s = as_w + as_{w+1}x + \dots + as_{n-1}x^w - as_0x^{w-1} - \dots - as_{w-1}x^{n-1}$.

c_1s can be represented as a vector: $(as_w, \dots, -as_{w-1})$. During the computation of \widehat{M} two cases are possible:

- $w < 2l_v$ then $\widehat{M} = c_2 - (c_1s)_{2l_v} = (\alpha_0 - as_w, \alpha_1 - as_{w+1}, \dots, \alpha_w + as_0, \dots, \alpha_{2l_v-1} + as_{(2l_v-1+w \bmod n)})$
- $w \geq 2l_v$ then $\widehat{M} = c_2 - (c_1s)_{2l_v} = (\alpha_0 - as_w, \alpha_1 - as_{w+1}, \dots, \alpha_{2l_v-1} - as_{(2l_v-1+w \bmod n)})$

Recall that since $s \leftarrow \psi_\sigma^n$, each of its coefficients belongs to $\{-1, 0, 1\}$. Let i be an integer such that $0 \leq i < l_v$ and $j \equiv i + w \pmod n$. For decryption there are the three following cases. (We cannot have the case where $\widehat{M}[i] = \alpha_i + as_j$ and $\widehat{M}[i + l_v] = \alpha_{i+l_v} - as_{j+l_v}$ because that implies $j + l_v \leq w + l_v$ and $w < j$ with $l_v > 0$ and $j \geq 0$.)

1. $\widehat{M}[i] = \alpha_i - as_j$ and $\widehat{M}[i + l_v] = \alpha_{i+l_v} - as_{j+l_v}$ If $\alpha_i = \frac{q}{2}$ one gets:
 - If $s_j = s_{j+l_v}$ or $s_j + s_{j+l_v} = -1$ we are in the Case 1 described in 4.1, where $\alpha_i - as_j = \widehat{M}[i]$. Then

$$\alpha_i = \alpha_{i+l_v} = \frac{q}{2}, \frac{(\alpha_i - as_j) + (\alpha_{i+l_v} - as_{j+l_v})}{2} = \begin{cases} \frac{q-2a}{2} & \text{if } s_j = s_{j+l_v} = 1 \\ \frac{q+2a}{2} & \text{if } s_j = s_{j+l_v} = -1 \\ \frac{q}{2} & \text{if } s_j = s_{j+l_v} = 0 \\ \frac{q+a}{2} & \text{if } s_j + s_{j+l_v} = -1 \end{cases}$$

These 3 values lie in $] \frac{q}{4}, \frac{3q}{4} [$ if $0 \leq a < \frac{q}{4}$.

- Otherwise we are in the Case 2 described in Paragraph 4.1, where $\alpha_i - as_j = \widehat{M}[i]$:

$$\alpha_i = \alpha_{i+l_v} = \frac{q}{2}, \frac{|(\alpha_i - as_j) - (\alpha_{i+l_v} - as_{j+l_v})|}{2} = \begin{cases} \frac{a}{2} & \text{if } s_j + s_{j+l_v} = 1 \\ a & \text{if } s_j = -1, s_{j+l_v} = 1 \\ & \text{or } s_j = 1, s_{j+l_v} = -1 \end{cases}$$

These values lie in $[0, \frac{q}{4} [$ if $0 \leq a < \frac{q}{4}$.

Then for both cases, if $c_1 = -ax^{n-w}$ with $\alpha_i, \alpha_{i+l_v} = \frac{q}{2}$, we can ensure that we have a 1 after comparison.

If $\alpha_i = 0$ then we are in the Case 1 described in Paragraph 4.1, where $\alpha_i - as_j = \widehat{M}[i]$:

$$\frac{(\alpha_i - as_j) + (\alpha_{i+l_v} - as_{j+l_v})}{2} = \begin{cases} a & \text{if } s_j = s_{j+l_v} = -1 \\ 0 & \text{if } s_j = -s_{j+l_v} \text{ or } s_j = s_{j+l_v} = 0 \\ -a & \text{if } s_j = s_{j+l_v} = 1 \\ \pm \frac{a}{2} & \text{otherwise} \end{cases}$$

Then these 3 values do not lie in $] \frac{q}{4}, \frac{3q}{4} [$ for $0 \leq a < \frac{q}{4}$.

2. $\widehat{M}[i] = \alpha_i + as_j$ and $\widehat{M}[i + l_v] = \alpha_{i+l_v} + as_{j+l_v}$. The proof is the same as above. We give here the different decryption cases:

- If $\alpha_i = \frac{q}{2}$ then two cases are possible: if $s_j = s_{j+l_v}$ or $s_j + s_{j+l_v} = 1$ then we are in the decryption Case 1 otherwise in the Case 2.
- If $\alpha_i = 0$ then we are in the decryption Case 1.

3. $\widehat{M}[i] = \alpha_i - as_j$ and $\widehat{M}[i + l_v] = \alpha_{i+l_v} + as_{j+l_v}$. The proof is the same as above. We give here the different decryption cases:

- If $\alpha_i = \frac{q}{2}$ then two cases are possible: if $s_j = -s_{j+l_v}$ or $s_j = 0, s_{j+l_v} = 1$ or $s_j = -1, s_{j+l_v} = 0$ then we are in the decryption Case 1, otherwise in the Case 2.
- If $\alpha_i = 0$ then we are in the decryption Case 1.

Proof of Proposition 4

Proof. According to Proposition 3 Eve can determine what Alice obtains at the end of the decryption procedure for every coefficient different from the key guesses. Assume that Eve wants to retrieve the w -th and $(w + l_v)$ -th coefficients of s . Let $\widehat{M} = c_2 - (c_1s)_{2l_v}$, due to $0 \leq w < (n - l_v)$ the only case to consider is $\widehat{M}[0] = as'_w - as_w$ and $\widehat{M}[l_v] = as'_{w+l_v} - as_{w+l_v}$.

Let $s'_w = s'_w = 1$ and $\frac{q}{8} < a < \frac{q}{4}$, so we are in the Case 1 described in Paragraph 4.1. Let see what happens with $\frac{\widehat{M}_0 + \widehat{M}_{l_v}}{2} = \frac{as'_w - as_w + as'_{w+l_v} - as_{w+l_v}}{2}$:

$$\frac{a - as_w + a - as_{w+l_v}}{2} = \begin{cases} 2a & \text{if } s_w = s_{w+l_v} = -1 \\ 0 & \text{if } s_w = s_{w+l_v} = 1 \\ \frac{3a}{2} & \text{if } s_w = 0, s_{w+l_v} = -1 \\ & \text{or } s_w = -1, s_{w+l_v} = 0 \\ \frac{a}{2} & \text{otherwise} \end{cases}$$

Then only the case $\frac{a - as_w + a - as_{w+l_v}}{2} = \frac{3a}{2}$ can put a 1 to \widehat{m}_0 if $\frac{q}{8} < a < \frac{q}{4}$.

With the same condition on a and with the same method Eve can have :

- If $s'_w = s'_{w+l_v} = 1$ and $\widehat{m}_0 = 1$ then $s_w = s_{w+l_v} = -1$
- If $s'_w = s'_{w+l_v} = -1$ and $\widehat{m}_0 = 1$ then $s_w = s_{w+l_v} = 1$
- If $s'_w = 1, s'_{w+l_v} = -1$ and $\widehat{m}_0 = 1$ then $s_w = -1$ and $s_{w+l_v} = 1$
- If $s'_w = -1, s'_{w+l_v} = 1$ and $\widehat{m}_0 = 1$ then $s_w = 1$ and $s_{w+l_v} = -1$

Proof of Proposition 5

Proof. Assume that Eve wants to retrieve the w -th and $(w + l_v)$ -th coefficients of s .

As Proof 5 the only case to consider is $\widehat{M}[0] = as'_w - as_w$ and $\widehat{M}[l_v] = as'_{w+l_v} -$

as_{w+l_v} . Suppose $\frac{q}{6} < a < \frac{q}{4}$, $s'_w = 1$ and $s'_{w+l_v} = 1$. Let see what happens with $\frac{\widehat{M}_0 + \widehat{M}_{l_v}}{2} = \frac{as'_w - as_w + as'_{w+l_v} - as_{w+l_v}}{2}$ (Case 1 described in Paragraph 4.1):

$$\frac{a - as_w + a - as_{w+l_v}}{2} = \begin{cases} \frac{3a}{2} & \text{if } s_w = -1 \text{ and } s_{w+l_v} = 0 \\ & \text{or } s_w = 0 \text{ and } s_{w+l_v} = -1 \\ \frac{a}{2} & \text{if } s_w = 0 \text{ and } s_{w+l_v} = 1 \\ & \text{or } s_w = 1 \text{ and } s_{w+l_v} = 0 \\ a & \text{if } s_w = s_{w+l_v} = 0 \end{cases}$$

With $\frac{q}{6} < a < \frac{q}{4}$ then only the case where the result is $\frac{3a}{2}$ can put a 1 to \widehat{m}_w . However Eve needs to determine if $s_w = -1$ or $s_{w+l_v} = -1$.

Suppose $a < \frac{q}{4}$, $s'_w = -1$ and $s'_{w+l_v} = 1$, $s_w = -1$ and $s_{w+l_v} = 0$ or $s_w = 0$ and $s_{w+l_v} = -1$. Here, we need to consider the both decryption cases described in Paragraph 4.1. Let see what happens:

- If $s_w = -1$ and $s_{w+l_v} = 0$ we are in Case 1 4.1 thus $\frac{q}{4} < a < \frac{3q}{4}$.
- If $s_w = 0$ and $s_{w+l_v} = -1$ we are in Case 2 4.1 thus $0 < \frac{|-a-2a|}{2} < \frac{q}{4}$ which implies $0 < a < \frac{3q}{8}$.

However $a < \frac{q}{4}$, then only one case can put a 1 to \widehat{m}_0 .

With the same condition on a and with the same method, Eve can retrieve the others values:

- If $s'_w = 1$, $s'_{w+l_v} = 1$ and $\widehat{m}_0 = 1$ then $s_w = -1$, $s_{w+l_v} = 0$ or $s_w = 0$, $s_{w+l_v} = -1$
 - If $s'_w = -1$, $s'_{w+l_v} = 1$ and $\widehat{m}_0 = 1$ then $s_w = 0$, $s_{w+l_v} = -1$ else $s_w = -1$, $s_{w+l_v} = 0$
- If $s'_w = -1$, $s'_{w+l_v} = -1$ and $\widehat{m}_0 = 1$ then $s_w = 1$, $s_{w+l_v} = 0$ or $s_w = 0$, $s_{w+l_v} = 1$
 - If $s'_w = 1$, $s'_{w+l_v} = -1$ and $\widehat{m}_0 = 1$ then $s_w = 0$, $s_{w+l_v} = 1$ else $s_w = 1$, $s_{w+l_v} = 0$

References

1. Alkim, E., et al.: NewHope (2019). <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions>
2. Avanzi, R., et al.: CRYSTALS-Kyber (2019). <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions>
3. B  etu, C., Durak, F.B., Huguenin-Dumittan, L., Talayhan, A., Vaudenay, S.: Misuse attacks on post-quantum cryptosystems. In: Ishai, Y., Rijmen, V. (eds.) EURO-CRYPT 2019. LNCS, vol. 11477, pp. 747–776. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_26

4. Bauer, A., Gilbert, H., Renault, G., Rossi, M.: Assessment of the key-reuse resilience of NewHope. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 272–292. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12612-4_14
5. Ding, J., Fluhrer, S., Rv, S.: Complete attack on RLWE Key exchange with reused keys, without signal leakage. In: Susilo, W., Yang, G. (eds.) ACISP 2018. LNCS, vol. 10946, pp. 467–486. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93638-3_27
6. Fluhrer, S.: Cryptanalysis of ring-LWE based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085 (2016). <https://eprint.iacr.org/2016/085>
7. 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
8. Guo, Q., Johansson, T., Yang, J.: A novel CCA attack using decryption errors against LAC. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 82–111. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_4
9. Hamburg, M.: Post-Quantum Cryptography Proposal: THREEBEARS (2019). <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions>
10. Huguenin-Dumittan, L., Vaudenay, S.: Classical misuse attacks on NIST round 2 PQC. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 2020. LNCS, vol. 12146, pp. 208–227. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57808-4_11
11. Liu, C., Zheng, Z., Zou, G.: Key Reuse Attack on NewHope Key Exchange Protocol. In: Lee, K. (ed.) ICISC 2018. LNCS, vol. 11396, pp. 163–176. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12146-4_11
12. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. *J. ACM (JACM)* **60**(6), 43 (2013)
13. Menezes, A., Ustaoglu, B.: On reusing ephemeral keys in Diffie-Hellman key agreement protocols. *IJACT* **2**(2), 154–158 (2010)
14. Montoya, S.: LAC attack. <https://github.com/ayotnomis/LACAttack>
15. Moody, D.: Post-Quantum Cryptography NIST’s plan for the future (2016). <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/pqcrypto-2016-presentation.pdf>
16. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical CCA2-Secure and Masked Ring-LWE Implementation. Cryptology ePrint Archive, Report 2016/1109 (2016). <https://eprint.iacr.org/2016/1109>
17. Qin, Y., Cheng, C., Ding, J.: An Efficient Key Mismatch Attack on the NIST Second Round Candidate Kyber. *IEEE* (2019)
18. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **41**(2), 303–332 (1999)
19. Xianhui, L., Yamin, L., Dingding, J., Haiyang, X., Jingnan, H., Zhenfei, Z.: LAC: lattice-based Cryptosystems (2019). <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions>



Enhancing Code Based Zero-Knowledge Proofs Using Rank Metric

Emanuele Bellini¹, Philippe Gaborit³, Alexandros Hasikos^{1,2(✉)},
and Victor Mateu¹

¹ Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, UAE
{emanuele.bellini, alexandros.hasikos, victor.mateu}@tii.ae

² Universitat Pompeu Fabra, Barcelona, Spain

³ University of Limogés, Limoges, France

gaborit@unilim.fr

Abstract. The advent of quantum computers is a threat to most currently deployed cryptographic primitives. Among these, zero-knowledge proofs play an important role, due to their numerous applications. The primitives and protocols presented in this work base their security on the difficulty of solving the Rank Syndrome Decoding (RSD) problem. This problem is believed to be hard even in the quantum model. We first present a perfectly binding commitment scheme. Using this scheme, we are able to build an interactive zero-knowledge proof to prove: the knowledge of a valid opening of a committed value, and that the valid openings of three committed values satisfy a given linear relation, and, more generally, any bitwise relation. With the above protocols it becomes possible to prove the relation of two committed values for an arbitrary circuit, with quasi-linear communication complexity and a soundness error of $2/3$. To our knowledge, this is the first quantum resistant zero-knowledge protocol for arbitrary circuits based on the RSD problem. An important contribution of this work is the selection of a set of parameters, and an a full implementation, both for our proposal in the rank metric and for the original LPN based one by Jain et al. in the Hamming metric, from which we took the inspiration. Beside demonstrating the practicality of both constructions, we provide evidence of the convenience of rank metric, by reporting performance benchmarks and a detailed comparison.

Keywords: Post Quantum · Code-based cryptography · Rank metric · Zero-knowledge proof · Identification protocol · Commitment scheme

1 Introduction

Due to the results of Grover [21] (1996) and Shor [33] (1997), the advancements in quantum information theory, and the discovery of new technologies, quantum computers are becoming more and more of a threat to the currently deployed cryptosystems, especially to those based on public key cryptography. Among these, *zero-knowledge proofs* (ZKP) are gaining particular attention due

to their numerous applications. They can be used to obtain identification and login mechanisms, cryptographic signature schemes, systems to enforce honest behaviour of the users, and to prove statements in public transaction systems such as blockchains. The growing interest from both academia and industry on the ZKP topic, has led to a series of results that improve upon previous theory and allow for the development of practical applications, and a standardization effort for zero-knowledge systems is also being carried on by the cryptographic community [34,37]. On the other hand, most of ZKP schemes are not quantum resistant.

Zero-knowledge proofs were first introduced by Goldwasser, Micali and Rackoff in 1989 [20]. In their work, they created a new proving procedure for communicating a proof, or in modern terms, an efficient *interactive proof system*. An interactive proof is a process in which a prover probabilistically convinces a verifier of the correctness of a mathematical proposition, also called statement. If the proof does not reveal to the verifier any additional information about the mathematical proposition, except if it is true or not, then it is called a *zero-knowledge proof*. A *zero-knowledge proof of knowledge* of a secret information is a special case of zero-knowledge proof, in which the statement consists only of the fact that the prover knows the secret information. Goldreich, Micali and Wigderson [19] showed how to make any proving system in NP (i.e. where the verifier is a deterministic, polynomial-time machine) zero knowledge, meaning that the verifier learns nothing but the correctness of the proposition. Furthermore, Impagliazzo and Yung in 1987 [22], and Ben-Or et al. in 1990 [8], showed that anything that can be proved by an interactive proof system can be proved with zero knowledge. Zero-knowledge proofs therefore provide complete privacy to the prover while convincing the verifier. Further research resulted in the study of non-interactive zero-knowledge proofs (NIZKs), a variant that does not require interaction between the prover and the verifier. Building on top of these, modern NIZK systems have become more efficient, including succinct proofs, sub-linear verifiers and highly efficient provers.

In this work, we will focus on quantum resistant interactive zero-knowledge proofs, with the property of public-coin, i.e. verifier's random coins are made public throughout the proof protocol. Notice that, a public-coin interactive proof of knowledge can always be converted into a non-interactive proof of knowledge by means of the Fiat-Shamir heuristic [14]. Furthermore, if the interactive proof is used as an identification tool, then the non-interactive version can be used directly as a digital signature.

1.1 Our Contribution

A commitment scheme is a cryptographic primitive that allows one to commit to a chosen value (or chosen statement) while keeping it hidden to others, with the ability to reveal (or to *open*) the committed value later. Commitment schemes are designed so that a party cannot change the value or statement after they have committed to it: that is, commitment schemes are binding.

In this work, we design and implement a perfectly binding and computationally hiding commitment scheme whose security relies on the hardness of solving the Rank Syndrome Decoding (RSD) problem, i.e. on the hardness of decoding random linear codes in the rank metric. This problem is believed to be hard even in the quantum model. Using this scheme, we are able to build an interactive zero-knowledge proof to prove: the knowledge of a valid opening of a committed value, and that the valid openings of three committed values satisfy a given linear relation, and, more generally, any bitwise relation.

With the above protocols it becomes possible to prove that the committed values c_0, c_1 satisfy $c_0 = C(c_1)$ for an arbitrary circuit C . As proved in [23], the total communication complexity of this protocol is $\mathcal{O}(|C|\mu \log \mu)$ where μ is the length of the committed messages. The soundness error is $2/3$, and thus for most applications must be lowered by (parallel) repetition.

Moreover, we also compute secure parameters, and implement¹ both schemes in the rank and Hamming metric, and compare their performances. Notice that, in [23], no parameters, nor an implementation was provided. Our proposal generates proofs that are 60% smaller and the size of the public parameters required is only a 1% with respect to the public parameters for the Hamming metric.

To our knowledge, this is the first zero-knowledge protocol for arbitrary circuits whose security relies on the difficulty of solving the Rank Syndrome Decoding problem, and the collision resistance of a hash function.

In Subsect. 1.2, we give an overview of the works related to our result. In Sect. 2, we introduce the basic notions needed to understand our scheme. In Sect. 3, we define a commitment scheme, and below it, in Sect. 4, we build our zero-knowledge protocols. In Sect. 5, we select a set of parameters both for our scheme and for its analogue in the Hamming metric, and we provide benchmarks of our implementations of the corresponding ZKP protocols. Finally, in Sect. 6, we draw the conclusions.

1.2 Related Works

This work is an adaptation of the protocols presented by Jain et al. in [23], where they show how to build a zero-knowledge protocol for arbitrary circuits reducing the security of their system to the difficulty of solving the Learning Parity with Noise problem, or, equivalently, to the difficulty of decoding a random linear code in the Hamming metric.

In turn, Jain's work is based on the preliminary identification protocol proposed by Stern in 1993 [35, 36], which inspired a long sequence of works improving either the scheme parameter size, or the communication cost. All the subsequent schemes derived from Stern's can be divided in four categories:

- **Type 1:** 3-pass protocols using the parity-check matrix of a code,
- **Type 2:** 3-pass protocols using the generator matrix of a code,

¹ A C++ implementation of the schemes described in this work can be found at https://github.com/ahasikos/rank_commitments.

- **Type 3:** 5-pass protocols using the parity-check matrix of a code,
- **Type 4:** 5-pass protocols using the generator matrix of a code.

Type 1 protocols can be seen as Zero-Knowledge Proof of Knowledge (ZKPoK) of a solution of an instance of the Syndrome Decoding problem for some specific code, where the syndrome is the public key and the corresponding error the private secret. As the original Stern proposal, they are 3-move Σ -protocols with a soundness error of $2/3$, and perfect completeness. The original Stern proposal used binary linear codes over the Hamming metric. Also, a second variant minimizing the computing load was presented, but its longer proof renders it unpractical. Double circulant codes, again in the Hamming metric, were proposed in 2007 by Gaborit and Girault in [16]. In 2011, Gaborit et al. adapted their proposal with double circulant codes to rank metric, obtaining the most compact code based identification scheme of Type 1. In 2008, Stern scheme was also adapted to the lattice setting by Kawachi et al. [24], who also extended the initial identification scheme to an *anonymous* identification scheme.

Using a generator matrix rather than the parity-check matrix, allows to reduce the communication cost, at expense of a slightly larger private key. This is why Type 2 protocols were introduced, in 1997, by Veron, in [38]. Type 2 protocols use a secret message and a secret error as the private key, and their encoding under a public generator matrix as the public key. Initially, the advantage in the communication cost was due to the fact that the committed value, which needs to be revealed in the response phase, was in the code plain message space rather than in the encoded message space. In 2012, Jain et al. [23] pointed out that Veron scheme did not reach perfect zero-knowledge, and proposed a variation of it, which they then used to construct zero-knowledge proof of knowledge of linear and multiplicative relations between committed messages. Jain version, though, lost the feature that was reducing the communication cost, as their commitment value was in the error space, which had the same size as the encoded message space. In 2018, Bellini et al. proposed the rank metric version of Veron scheme, thought without providing a security proof, and their scheme was attacked in 2019 in [25]. This is, so far, the only Stern-based scheme that has been attacked.

Notice that Type 1 and Type 2 protocols are 3-pass Σ -protocols, with perfect completeness and a soundness error (often referred to as cheating probability) of $2/3$. Type 3 and Type 4 protocols were introduced to reduce the soundness error from $2/3$ to almost $1/2$, by performing 5 steps instead of 3. This allows to run less parallel execution of the protocol to reach a smaller desired soundness error, and, sometimes, a smaller communication cost at expense of some extra computation.

The first Type 3 protocol was presented in the second variant of Stern's original proposal. However, also this alternative had a larger proof and was not practical. In 2010, Cayrel-Veron-El Yousfi Alaoui (CVE) [12] presented a 5-pass identification protocol with soundness error of $q/(2q - 2)$, using codes over \mathbb{F}_q^m , this time improving significantly the communication cost compared to the initial 5-pass proposal by Stern. A version of CVE scheme in the rank metric is presented in [7], though lacking a security proof. It is worth noting that

the parameters proposed for this particular rank version of CVE scheme do not improve key size nor communication cost with respect to the Hamming metric version. A lattice based version of CVE was presented in 2012 by Cayrel et al. [11], reaching a smaller public key, but larger private key and communication cost than CVE. This scheme also improves under all aspects the Type 1 lattice-based scheme of Kawachi et al. [24]

The first Type 4 protocol was presented in 2011 in [2] by Aguilar et al., where double circulant codes were used. The key size, the communication cost and the soundness error of this protocol were later significantly improved in 2019, by Bellini et al. in [6], by replacing the Hamming metric with the rank metric. A lattice based version of the Jain et al. protocol was presented by Martínez and Morillo in 2019 [29], where they also use some ideas from [26] and [40].

Table 1. Summary of Stern-like protocols.

Name	Ref	Year	Metric	Setting	Aim	Notes
3-pass, with parity-check matrix						
Stern(1)	[35, 36]	1993	Hamm	Linear codes	Identification	-
Stern(2)	[35, 36]	1993	Hamm	Linear codes	Identification	Minimize computing load, proof not practical
GG	[16]	2007	Hamm	Double Circulant codes	Identification	-
KTX	[24]	2008	Euclidean	Lattices	Anonymous Identification	-
GSZ	[18]	2011	Rank	Double Circulant codes	Identification	-
3-pass, with generator matrix						
Veron	[38]	1997	Hamm	Linear codes	Identification	Not perfect ZK
JKPT	[23]	2012	Hamm	Linear codes	ZKPoK of relations	-
BKLP	[9]	2015	Euclidean	Lattices	ZKPoK of relations	-
BCHMM	[7]	2018	Rank	Linear Codes	Signature	Attacked in [25]
This work	-	-	Rank	Linear codes	ZKPoK of relations	-
5-pass, with parity-check matrix						
Stern(3)	[35, 36]	1993	Hamm	Linear codes	Identification	Proof not practical
CVE	[12]	2010	Hamm	q -ary Linear Code	Identification	-
CLRS	[11]	2012	Euclidean	Lattices	Identification	-
BCHMM	[7]	2018	Rank	Linear Codes	Signature	-
5-pass, with generator matrix						
AGS	[2]	2011	Hamm	Double Circulant codes	Identification	-
BCGMM	[6]	2019	Rank	Double Circulant codes	Signature	-
MM	[29]	2019	Euclidean	Ideal Lattices	ZKPoK of relations	-

The authors do not propose a set of parameters and leave as future work an implementation of their scheme.

All the above mentioned protocols are believed to be secure even against quantum adversaries, thought the situation is more uncertain as far as it concern the analogue of the Fiat-Shamir transform for 5-pass protocols.

In the case of lattices, it is possible to construct zero-knowledge proofs using approaches different from Stern, as it was done, for example, in [26,28,31]. A summary of the above described Stern-like protocols can be found in Table 1.

2 Preliminaries and Notations

2.1 Codes in the Rank Metric

We use $\mathcal{M}_{r,c}(R)$ and $\mathcal{M}_{r,c}^*(R)$ to denote, respectively, the set of all matrices and the set of all full rank matrices with r rows and c columns with entries over the ring R . Given $M_1 \in \mathcal{M}_{r,c_1}(R)$ and $M_2 \in \mathcal{M}_{r,c_2}(R)$, we indicate with $M_1 \| M_2$ the concatenation of the two matrices.

A linear $(n, k)_q$ -code C is a vector subspace of $(\mathbb{F}_q)^n$ of dimension k , where k and n are positive integers such that $k < n$, q is a prime power, and \mathbb{F}_q is the finite field with q elements. Elements of the vector space are called vectors or words, while elements of the code are called codewords. A matrix $G \in \mathcal{M}_{k,n}^*(\mathbb{F}_q)$ is called a generator matrix of C if its rows form a basis of C , i.e. $C = \{x \cdot G : x \in (\mathbb{F}_q)^k\}$. A matrix $H \in \mathcal{M}_{n-k,n}^*(\mathbb{F}_q)$ is called a parity-check matrix of C if $C = \{x \in (\mathbb{F}_q)^n : H \cdot x^T = 0\}$.

In this paper, we work with codes in the *rank metric*. Given a fixed basis $\beta = \{\beta_1, \dots, \beta_m\}$ of $(\mathbb{F}_q)^m$, a vector $a \in (\mathbb{F}_q^m)^n$ can be represented as a matrix with entries in \mathbb{F}_q , by expanding each component of a_i with respect to β in a column $(a_{1,i}, \dots, a_{m,i})^T$, where $a_i = \sum_{j=1}^m a_{j,i} \beta_j, i = 1, \dots, n$. We define the rank $w_R(v)$ of a vector v as the rank of its *matrix representation*, with respect to β . We denote the previous matrix representation as $\phi_\beta(v)$, and by ϕ_β^{-1} the inverse map. In what follows, we will omit β as we consider it fixed.

To send a binary vector of a certain Hamming weight to *any* other vector of the same Hamming weight, it is sufficient to apply a random permutation to vector components. The map with the analogue property in the rank metric, i.e. sending a vector of a certain rank to *any* other vector of the same rank, can be defined as follows (see [18]).

Definition 1. Let $Q \in \mathcal{M}_{m,m}^*(\mathbb{F}_q)$ be a q -ary matrix of size $m \times m$, $P \in \mathcal{M}_{n,n}^*(\mathbb{F}_q)$ be a q -ary matrix of size $n \times n$, and $v = (v_1, \dots, v_n) \in (\mathbb{F}_q^m)^n$. We define the function $\Pi_{P,Q}$ such that $(\pi_1, \dots, \pi_n) = \Pi_{P,Q}(v) = \phi^{-1}(Q \cdot \phi(v) \cdot P) \in (\mathbb{F}_q^m)^n$, where for $h = 1, \dots, n$, $\pi_h := \beta_1 \sum_{i=1}^m \sum_{j=1}^n Q_{1,i} v_{i,j} P_{j,h} + \dots + \beta_m \sum_{i=1}^m \sum_{j=1}^n Q_{m,i} v_{i,j} P_{j,h}$, with $\beta = \{\beta_1, \dots, \beta_m\}$ a basis of $(\mathbb{F}_q)^m$.

In [18], it is proved that, for any $x, y \in (\mathbb{F}_q^m)^n$, and any full rank $P \in \mathcal{M}_{n,n}^*(\mathbb{F}_q)$ and any full rank $Q \in \mathcal{M}_{m,m}^*(\mathbb{F}_q)$, then $\Pi_{P,Q}$ has the rank preserving property, i.e. $w_R(\Pi_{P,Q}(x)) = w_R(x)$, and is a linear mapping, i.e. $a\Pi_{P,Q}(x) + b\Pi_{P,Q}(y) =$

$\Pi_{P,Q}(ax + by)$. Furthermore, for any $x, y \in (\mathbb{F}_{q^m})^n$ such that $w_R(x) = w_R(y)$, it is possible to find $P \in \mathcal{M}_{n,n}^*(\mathbb{F}_q)$ and $Q \in \mathcal{M}_{m,m}^*(\mathbb{F}_q)$ such that $x = \Pi_{P,Q}(y)$. The last property shows that, given a vector of a certain rank, it is possible to associate to it any other vector of the same rank by modifying P and Q . This property will be used in the zero-knowledge proof of the proposed scheme. Notice also that $\Pi_{P,Q}$ is invertible if P and Q are.

We denote by $\binom{n}{s} = \prod_{i=0}^{s-1} \frac{q^n - q^i}{q^s - q^i}$ the number of s -dimensional vector subspaces of $(\mathbb{F}_q)^n$ over \mathbb{F}_q . A ball $B_R^r(a)$ in the rank metric of radius r centered in a vector $a \in (\mathbb{F}_{q^m})^n$ is the set of all vectors in rank distance at most r from a . It can be shown [39] that $|B_R^r(a)| = \sum_{i=1}^r \binom{m}{i} \prod_{j=0}^{i-1} (q^n - q^j)$, which does not depend on a .

The following bound plays an important role in the choice of the parameters of our schemes.

Theorem 1 (*q -ary Gilbert-Varshamov Bound in rank metric [15]*). *Let $A_{q^m}^R(n, d)$ be the maximum cardinality of a linear block code over \mathbb{F}_{q^m} of length n , size M , and minimum distance d in the rank metric. Then $A_{q^m}^R(n, d) \geq \frac{q^{mn}}{|B_R^{d-1}(0)|}$.*

Both in the Hamming and in the rank metric, random codes over \mathbb{F}_q asymptotically achieve the Gilbert-Varshamov bound [15]. Furthermore, they have close to optimal correction capability [27]. This result is important for the scheme that we propose, as it allows to choose random generator (or parity-check) matrices as long as the code parameters respect the bound.

2.2 Rank Decoding Problem

We now define the problem upon which the security of the commitment schemes we present is based. This problem is equivalent to the *decoding problem* for random linear codes, which consists of searching for the closest codeword to a given vector. More precisely, given $G, y = xG + e$, and the weight w , the decoding problem consists in finding the pair (x, e) , where the weight of e is w . In the case of linear codes, it can be easily shown that the decoding problem is equivalent to the problem in which the syndrome $s = Hy$ of the received vector is given instead of the received vector itself. In this case we use the term Syndrome Decoding (SD) when referring to linear code in the Hamming metric, and Rank Syndrome Decoding (RSD) when referring to linear code in the rank metric.

Definition 2 (RSD Distribution). *Given the positive integers n, k , and ρ , the $RSD(n, k, \rho)$ Distribution chooses $H \leftarrow_s \mathcal{M}_{n-k,n}^*(\mathbb{F}_{q^m})$ and $x \leftarrow_s (\mathbb{F}_{q^m})^n$ such that $w_R(x) = \rho$, and outputs $(H, H \cdot x^T)$*

Problem 1 (RSD Problem). On input $(H, y^T) \in \mathcal{M}_{n-k,n}^*(\mathbb{F}_{q^m}) \times (\mathbb{F}_{q^m})^{n-k}$ from the RSD distribution, the Rank Syndrome Decoding problem $RSD(n, k, \rho)$ asks to find $x \in (\mathbb{F}_{q^m})^n$ such that $H \cdot x^T = y^T$ and $w_R(x) = \rho$.

The previous problem can be defined correspondingly also in the Hamming metric, in which setting the problem has been proven to be NP-complete [10]. The

RSD problem has recently been proven difficult with a probabilistic reduction to the Hamming scenario in [1]. By applying the transformation described in [1] it can be shown that the Decisional version of the RSD problem can be reduced to a search problem for the Hamming metric, providing some evidence on the hardness of the problem.

2.3 Commitment Schemes

In this section we define a commitment scheme and the properties which are related to this paper.

Definition 3. *A triple of algorithms (Setup,Com,Ver) is called a commitment scheme if it satisfies the following:*

- On input 1^λ , the setup algorithm Setup outputs the public commitment parameters pp .
- The commitment algorithm Com takes as inputs a message m from a message space M and a the the public commitment parameters pp , and outputs a commitment/opening pair (c, d) .
- The verification algorithm Ver takes the parameters pp , a message m , a commitment c and an opening d and outputs true or false.

The commitment scheme we describe satisfies these security properties:

- *Correctness:* Ver evaluates to true if the inputs are honestly computed, i.e.,

$$\Pr[\text{Ver}(\text{pp}, m, c, d) = \text{true}; \text{pp} \leftarrow_s \text{Setup}(1^\lambda), m \in M, (c, d) \leftarrow_s \text{Com}(m, \text{pp})] = 1$$

- *Perfect binding:* With overwhelming probability over the choice of the public commitment parameters $\text{pp} \leftarrow_s \text{Setup}(1^\lambda)$, no commitment c can be opened in two different ways, i.e.,

$$(\text{Ver}(\text{pp}, m, c, d) = \text{true}) \text{ and } (\text{Ver}(\text{pp}, m', c, d') = \text{true}) \implies m = m'$$

- *Computational hiding:* A commitment c computationally hides the committed message if, with overwhelming probability over the choice of the value $\text{pp} \leftarrow_s \text{Setup}(1^\lambda)$, for every $m, m' \in M$, and for $(c, d) \leftarrow_s \text{Com}(m, \text{pp})$ and $(c', d') \leftarrow_s \text{Com}(m', \text{pp})$ the distributions c and c' are computationally indistinguishable.

2.4 Zero-Knowledge Proof of Knowledge

A zero-knowledge proof of knowledge is a protocol in which P wants to prove to a V the knowledge of some secret information without revealing anything about it, except the fact that he knows it. More formally, in a zero-knowledge proof for a binary relation R , the two parties have a common input y and P has a private input w such that $(y, w) \in R$. To be defined as zero-knowledge, the protocol must then satisfy the following three properties:

- *Completeness*: for an honest prover, the verifier always accepts.
- *Zero-knowledge*: for every potentially malicious verifier V' there exists a PPT simulator only taking y as an input whose output is indistinguishable from conversations of V' with an honest prover.
- *Proof of knowledge*: from every prover P which can make the verifier accept with a probability larger than a threshold k (the *knowledge error*), a w' satisfying $(y, w') \in R$ can be extracted efficiently in a rewindable black-box way.

For a more formal definition we refer to Bellare and Goldreich [5].

3 A Commitment Scheme in the Rank Metric

In this section we describe a perfectly binding commitment scheme whose security depends on the difficulty of solving the Rank Syndrome Decoding (RSD) problem. This commitment scheme follows the structure of the commitment scheme presented [23], based on the LPN problem.

The scheme is parameterized by the following values: the prime characteristic q (in our implementation we set $q = 2$) and the degree m of a q -ary extension field \mathbb{F}_{q^m} , the bit length μ of a message $m \in \mathbb{F}_q^\mu$, the bit length π of the randomness $s \in \mathbb{F}_q^\pi$, the length n of the linear code C , and the rank weight ρ of an error $e \in \mathbb{F}_{q^m}^n$. The dimension k of the code C is given by $k = (\mu + \pi)/m$ (we require μ and π to be both multiples of m , so that $(s||m)$ can be seen as an element of $\mathbb{F}_{q^m}^k$). Notice also that an instance of the RSD problem is hard if the weight ρ is taken close to the Gilbert-Varshamov bound. Once the scheme public parameters q, m, μ, π, n, ρ are chosen accordingly with the security parameter λ (see Subsect. 5.1 for an example of actual values), then the commitment scheme is defined by the following three algorithms (**Setup**, **Com**, **Ver**):

Setup(1^λ)	Com $_G(m)$	Ver $_G(c, m', s')$
$G_m \leftarrow_s \mathcal{M}_{\frac{\mu}{m}, n}^*(\mathbb{F}_{q^m})$	$s \leftarrow_s \mathbb{F}_2^\pi$	$e' = c + (s' m') \cdot G$
$G_s \leftarrow_s \mathcal{M}_{\frac{\pi}{m}, n}^*(\mathbb{F}_{q^m})$	$e \leftarrow_s \mathbb{F}_{q^m}^n, \text{ s.t. } \text{wr}(e) = \rho$	if $\text{wr}(e') = \rho$ return True
return $G = (G_s^\top G_m^\top)^\top$	$c = (s m) \cdot G + e$	else return False
	return c, s	

The matrix G is called the *public commitment key*. We will write **Com** and **Ver**, omitting G , when clear from the context. The second output s of the **Com** algorithm is needed by the party generating the commitment, in order to prove that it was the one generating the commitment.

Theorem 2. *Let us fix q, m, μ, π, n, ρ so that the RSD problem is hard. Let $G \in \mathcal{M}_{k, n}^*(\mathbb{F}_{q^m})$ be the generator matrix of a random linear code C of dimension k and length n . Then the above defined commitment scheme is perfectly binding and computationally hiding.*

Proof. We first prove that the scheme is perfectly binding. First, let us recall that random linear codes over \mathbb{F}_{q^m} asymptotically achieve the Gilbert-Varshamov bound. Thus, with overwhelming probability, the code C has minimum rank distance greater than $d_C = 2\rho$. This means that no codeword in C can have rank weight less than or equal to d_C . Now, let us assume, by contraposition, that there exists two different openings \mathbf{m}, \mathbf{m}' for a commitment \mathbf{c} . This means that $\mathbf{e} = \mathbf{c} + (\mathbf{s}||\mathbf{m}) \cdot G$ and $\mathbf{e}' = \mathbf{c} + (\mathbf{s}'||\mathbf{m}') \cdot G$ are such that $w_R(\mathbf{e}) = w_R(\mathbf{e}') = \rho$. Since $\mathbf{e} + \mathbf{e}' = ((\mathbf{s}||\mathbf{m}) + (\mathbf{s}'||\mathbf{m}')) \cdot G \in C$, and because of the metric properties, we have that $w_R(\mathbf{e} + \mathbf{e}') \leq w_R(\mathbf{e}) + w_R(\mathbf{e}') = 2\rho = d_C$. This means that the codeword $(\mathbf{e} + \mathbf{e}')$ has minimum rank weight less than the code distance. Since this is impossible, than \mathbf{m} must be different from \mathbf{m}' .

To prove that the scheme is computationally hiding, we first notice that $\mathbf{c} = \mathbf{s} \cdot G_s + \mathbf{m} \cdot G_m + \mathbf{e}$. Then we conclude that \mathbf{c} is indistinguishable from a random vector of the same length, since both \mathbf{s} and \mathbf{e} are sampled from a random distribution, and we are assuming that $\mathbf{s} \cdot G_s + \mathbf{e}$ is also indistinguishable from random. \square

4 Zero Knowledge Proof Protocols

In this section we describe three Σ -protocol. The first protocol is a proof of knowledge of a valid opening. It is a variant of Stern protocol [35], or, more precisely, of the dual of it due to Veron [38]. The second protocol allows to prove that committed strings satisfy any linear relation. Finally, the third protocol allows to prove that committed strings satisfy any bitwise relations, as bitwise AND, NAND, OR, or NOR. Since NAND is functionally complete, using this protocol we can construct Σ -protocols for any relation amongst committed messages. For all three protocols, we follow the ideas and proofs of [23], and adapt them to rank metric.

4.1 Proving Knowledge of a Valid Opening

The following Σ -protocol proves knowledge of a valid opening for commitments of the form described in Sect. 3, i.e., it shows possession of $\mathbf{s}, \mathbf{m}, \mathbf{e}$ such that $\mathbf{y} = (\mathbf{s}||\mathbf{m}) \cdot G + \mathbf{e}$ for an error satisfying $w_R(\mathbf{e}) = \rho$. For the sake of notation convenience, we will sometimes write \mathbf{x} to denote the vector $(\mathbf{s}||\mathbf{m})$. The protocol is described in Fig. 1. The inputs for P are $\mathbf{x} \in (\mathbb{F}_{q^m})^k$ and $\mathbf{e} \in (\mathbb{F}_{q^m})^n$ s.t. $w_R(\mathbf{e}) = \rho$. The pair (\mathbf{x}, \mathbf{e}) is the secret P wants to prove the knowledge of. Both P and V share as input the public parameters: the generator matrix G and the error rank weight ρ . The function $E()$ takes a sequence of inputs and converts it to a binary string of size μ (a collision resistant hash or XOF function can be used), suitable to be used as input message for the Com or Ver algorithm. Notice that, the protocol uses Π as defined in Subsect. 2.1 with P and Q being invertible. This allows us to operate with f and $f + \mathbf{e}$ in a way that preserves the rank of the error but still hides it. The Π operation preserves linearity and is the key on the adaptation from Hamming to rank metric.

Theorem 3. *The protocol described in Fig. 1 is a Σ -protocol for the following relation: $\mathcal{R}_{RSD} = \{((G, y), (s, m, e)) : y = (s||m) \cdot G + e \text{ and } w_R(e) = \rho\}$*

The proof of Theorem 3 can be found in Sect. B.

4.2 Proving Linear Relations

As it is introduced in Jain et al. paper, our adaptation into rank metric is also suitable to prove linear relations of several valid openings. The main idea is to provide a method by which a prover P can prove knowledge of a bitwise relation between the committed messages without showing the messages. The whole construction is similar in the sense that the relation is still holding in the message space of the commitments.

Given the three q -ary vectors m_1, m_2, m_3 and two q -ary matrices $X_1, X_2 \in \mathcal{M}_{\mu, \mu}(\mathbb{F}_q)$ such that $m_3 = X_1 m_1 + X_2 m_2$, a prover can prove in zero knowledge the existence of this relation by running the protocol detailed in Fig. 2. P first commits to the values obtaining $y_i = (s_i || m_i)G + e_i$, and then generates v_1 and

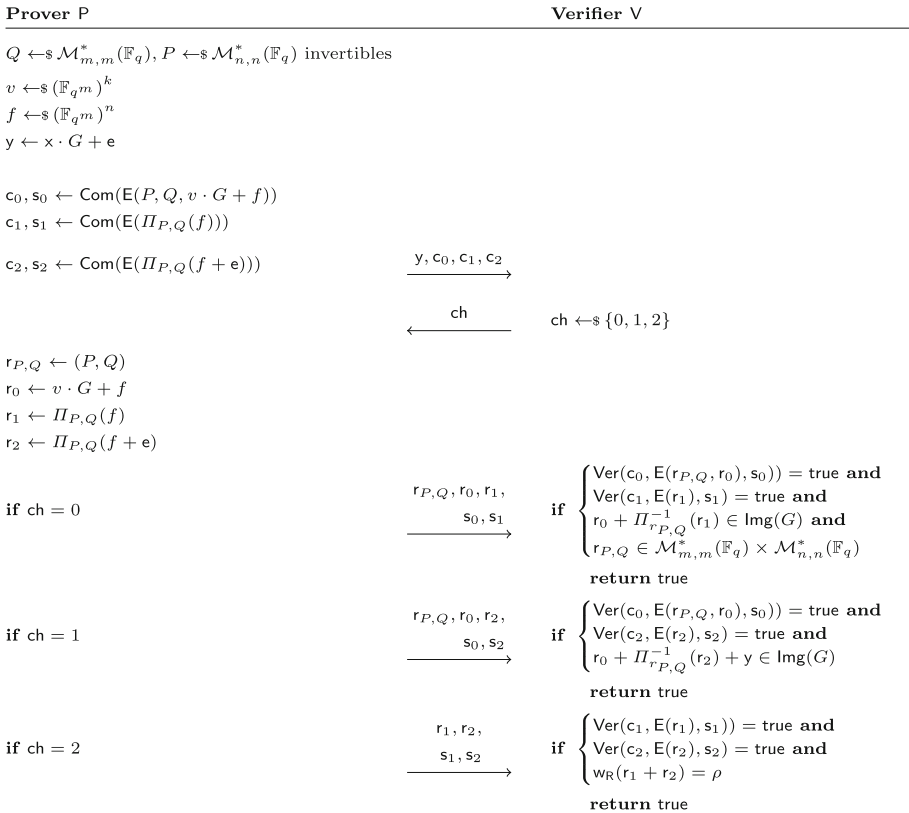


Fig. 1. A Σ -protocol proving valid opening of a commitment in the rank metric.

v_2 at random to later compute $v_3 = X_1v_1 + X_2v_2$. With this second set of values sharing the same linear relations the prover proceeds by proving valid opening of the v_i values using the proof from Subsect. 4.1 but now the verifier validates different computations regarding the linear relation and how it applies to either v_i or $v_i + m_i$ depending on the challenge. The protocol protects the values m_i by masking them with the random values v_i which, given that they share the linear relations, can be evaluated without disclosing their values. It is worth noting that, both prover P and verifier V know the public parameters y_1, y_2, y_3, G, ρ , and the relations X_1 and X_2 . On the other hand, only the prover P knows $x = (s_i || m_i)$ and e .

4.3 Proving Multiplicative Relations

When the properties we want to prove are multiplicative such as $m_3 = m_1 \circ m_2$, we will follow the original idea and try to reduce the multiplicative relation into a linear relation in order to use the construction from Subsect. 4.2. In a nutshell, the prover P will have the commitments y_1, y_2 , and y_3 of the messages m_1, m_2 ,

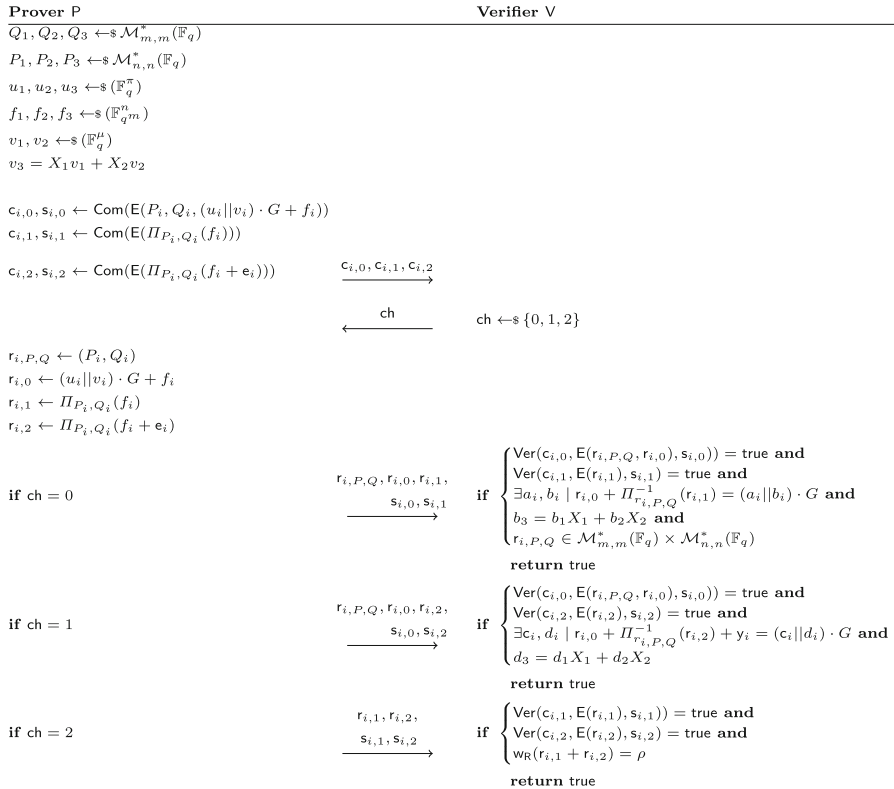


Fig. 2. A Σ -protocol proving linear relations of valid opening in the rank metric.

and m_3 . In order to prove the \circ relation, P will begin sampling vectors m'_i sharing the same multiplicative relation and adding restrictions to its structure. After this, P will generate a random permutation matrix R such that $R \cdot m'_i = m_i$. Finally, P will use the proof of linear relation with the linear relation R but, given that R is not known by V, it will send a commitment to R and also commitments to m'_i for $i = 1, 2, 3$. The detailed version of the protocol is presented in Fig. 3 (commitment) and Fig. 4 (challenge and response). The inputs for P are $m_i \in \mathbb{F}_2^\mu$, $e_i \in (\mathbb{F}_{q^m})^n$ s.t. $w_R(e_i) = \rho$, and $s_i \in \mathbb{F}_2^\pi$ for $i = 1, 2, 3$. Both P and V share as input the public parameters: the generator matrix G , the error rank weight ρ , and the commitments y_i for $i = 1, 2, 3$. Besides this, they also share knowledge of the multiplicative relation \circ . For the purpose of readability, we use similar notation to the original Jain et al. protocol, which includes the use of m_i^j to denote the j -th bit block of m_i . Following this reasoning, R^j would be the submatrix resulting from taking only the columns from $(j - 1)\mu m + 1$ to $j\mu m$. We use the same notation for Q_i, Q'_i, P_i , and P'_i . Notice that, the conversion from Hamming to rank metric, is again made possible by the use of the functions $\Pi_{P,Q}$, which are linear mappings preserving the rank.

5 Implementation

In the original proposal from Jain et al. [23], no set of parameters was provided, and consequently no implementation to prove the efficiency of the scheme in a real scenario. In this section, we first fix a set of parameters for a quantum security level of 128 bit, and then we provide benchmarks of our implementation of the commitment schemes in the Hamming and the rank metric.

5.1 Parameters

For the Jain et al. commitment scheme we have to choose a proper set of parameters n, k, w such that the syndrome decoding problem in the Hamming metric can be solved with at least 2^{128} operations with a quantum or a classical computer. The difficulty of solving the syndrome decoding problem in the Hamming metric and in the *full distance decoding* scenario² and when $n \approx 2k$ (the hardest case), is given by $2^{0.097n}$ [30]. For quantum security, the exponent should be divided by two. To obtain a security level of λ bit in the quantum scenario, then $n = 2\lambda/0.097$. For $\lambda = 128$ we obtain $n = 2640, k = 1320$. Since the Gilbert-Varshamov bound for the given n, k is $d = 294$, we choose w close to this value, e.g. $w = 284$. We recall that, in [23], the dimension k is split in two values ℓ and v (in this paper corresponding to π and μ , respectively), where ℓ is the security parameter, resulting in $\ell = 128$ and $v = 1192$.

² In the full distance decoding, the attacker receives an arbitrary point and aims at decoding the closest codeword. In the *half distance decoding*, the attacker knows that the error vector is within the error correction distance, i.e. $w_H(e) \leq \lfloor (d - 1)/2 \rfloor$. In this case the decoding complexity is $2^{0.0473n}$.

Prover P
Verifier V

$m'_1, m'_2, m'_3 \leftarrow \mathbb{F}_2^{4\mu}$ such that
 $m'_3 = m'_1 \circ m'_2$
 $\forall (a, b) \in (\mathbb{F}_q)^2, \#[(m'_1[j], m'_2[j]) = (a, b)] = \mu$
 $R \leftarrow \mathcal{M}_{\mu, 4\mu}(\mathbb{F}_q)$ s.t. $R \cdot m'_i = \mathbf{m}_i$ and $\text{Rank}(R) = \mu$
for $i = 1, 2, 3$
 for $j = 1, 2, 3, 4$
 $s'^j_i \leftarrow \mathbb{F}_q^\pi$
 $e'^j_i \leftarrow \mathbb{F}_{2^m}^n$ s.t. $\text{wr}(e'^j_i) = \rho$
 $y'^j_i = (s'^j_i || m'^j_i) \cdot G + e'^j_i$
 $Q'^j_i \leftarrow \mathcal{M}_{m, m}^*(\mathbb{F}_2)$
 $P'^j_i \leftarrow \mathcal{M}_{n, n}^*(\mathbb{F}_2)$
 $v'^j_i \leftarrow \mathbb{F}_2^\mu$
 $u'^j_i \leftarrow \mathbb{F}_2^\pi$
 $f'^j_i \leftarrow \mathbb{F}_{2^m}^n$
 $c'^j_{i,0}, s'^j_{i,0} \leftarrow \text{Com}(E(P'^j_i, Q'^j_i, (u'^j_i || v'^j_i) \cdot G + f'^j_i))$
 $c'^j_{i,1}, s'^j_{i,1} \leftarrow \text{Com}(E(\Pi_{P'^j_i, Q'^j_i}(f'^j_i)))$
 $c'^j_{i,2}, s'^j_{i,2} \leftarrow \text{Com}(E(\Pi_{P'^j_i, Q'^j_i}(f'^j_i + e'^j_i)))$
 endfor
 $v_i = \sum_{j=1}^4 R^j \cdot v'^j_i$
 $Q_i, \leftarrow \mathcal{M}_{m, m}^*(\mathbb{F}_2)$
 $P_i, \leftarrow \mathcal{M}_{n, n}^*(\mathbb{F}_2)$
 $u_i, \leftarrow \mathbb{F}_2^\pi$
 $f_i, \leftarrow \mathbb{F}_{2^m}^n$
 $c_{i,0}, s_{i,0} \leftarrow \text{Com}(E(P_i, Q_i, (u_i || v_i) \cdot G + f_i))$
 $c_{i,1}, s_{i,1} \leftarrow \text{Com}(E(\Pi_{P_i, Q_i}(f_i)))$
 $c_{i,2}, s_{i,2} \leftarrow \text{Com}(E(\Pi_{P_i, Q_i}(f_i + e_i)))$
 endfor
 $c, s \leftarrow \text{Com}(E(y'_1, y'_2, y'_3))$
 $c_R, s_R \leftarrow \text{Com}(E(R))$

$$\begin{array}{c}
 c, c_R, c_{i,0}, c_{i,1}, c_{i,2} \\
 \xrightarrow{\hspace{1.5cm}} c'_{i,0}, c'_{i,1}, c'_{i,2}
 \end{array}$$

Fig. 3. Commitment step of the Σ -protocol proving multiplicative relations in the rank metric.



Fig. 4. Challenge and response steps of the Σ -protocol proving multiplicative relations in the rank metric.

For our commitment scheme, we choose a proper set of parameters q, m, n, k, r such that the SD problem in the rank metric can not be solved with less than 2^{128} operations using a quantum or a classical computer. To obtain a security level of $\lambda = 128$ bit in the quantum scenario, we selected the following parameters: $q = 2, m = 43, n = 38, k = 17, \rho = 13$. Notice that the Gilbert-Varshamov bound for the given q, m, n, k is $d = 15$. Finally, we have $\pi = 129$ (greater than $\lambda = 128$) and $\mu = mk - \pi = 602$. The work factor (i.e. the base 2 logarithm of the attack time complexity) of the known attacks for the chosen parameters is summarized in Table 2. Since the cheating probability of the scheme is $2/3$, to reach 128 bit of security, we need to repeat the protocol δ times, where δ is the least integer such that $(2/3)^\delta < 2^{-128}$. This gives us $\delta = 219$.

Table 2. Work factor of the known attacks to the rank syndrome decoding problem for $q = 2, m = 43, n = 38, k = 17, \rho = 13$.

Reference	Attack type	Complexity	Work factor
[13]	Combinatorial	$(n\rho + m)^3 q^{(m-\rho)(\rho-1)/2}$	207.21
[32] (1)	Combinatorial	$(m\rho)^3 q^{(\rho-1)(k+1)/2}$	135.38
[32] (2)	Combinatorial	$(k + \rho)^3 \rho^3 q^{(m-\rho)(\rho-1)/2}$	205.82
[17] (1)	Combinatorial	$(n - k)^3 m^3 q^{\rho \lfloor k * m / (2n) \rfloor}$	146.46
[17] (2)	Combinatorial	$(n - k)^3 m^3 q^{(\rho-1) \lfloor (k-1)m / (2n) \rfloor}$	137.46
[3]	Combinatorial	$(n - k)^3 m^3 q^{\rho \lceil (k+1)m / (2n) \rceil - m}$	129.46
[17] (3)	Algebraic	$\rho^3 k^3 q^{\rho \lceil ((\rho+1)(k+1) - (n+1)) / \rho \rceil}$	244.36
[4]	Algebraic	$\left(\frac{\binom{(m+n)\rho}{\rho+1}}{(\rho+1)!} \right)^w, w = 2.807$	292.55

5.2 Sizes and Communication Cost Comparison

Table 3 shows a comparison of the secret and public parameter sizes and the average communication cost of one round of the Σ -protocol of Fig. 1, for a quantum security level of 128 bits, for both Hamming and rank metric. In the rank metric, the average communication cost is about 60% lower, while the public parameters size is two orders of magnitude smaller. However, the size of the secret in the ZKP is also 40% smaller. The size of the secret can be evaluated as both a benefit and a drawback. On one side, the proof is limited on the size of the secret, therefore, bigger secrets would also require bigger proofs. On the other side, this also means the proof is able to provide security to a smaller value. A common application of this last argument is a signature scheme, where the secret is the private key of the signer. In that case, the size of the private key could

be smaller and, therefore, the scheme would be more efficient in terms of size. Notice that the communication cost could be reduced by using techniques similar to the ones presented in [6]. Also, secret and public parameters sizes could be reduced in both Hamming and rank metric by using ideal or quasi-cyclic codes instead of random linear codes.

Table 3. Communication cost and parameters bit sizes of the Σ -protocol of Fig. 1 for a quantum security level of 128 bits.

		Parameters	Secret	Public Param.	Average communication
Hamming [23]	Formula	(n, k, w)	$k + n$	$n + kn + \log_2(w)$	$5n + \lceil 2/3(n \log_2(n)) \rceil + 2\lambda$
	Bits	(2640,1320,284)	3960	3487449	33461
Rank (this work)	Formula	(q, m, n, k, ρ)	$mk + mn$	$mn + mkn + \log_2(\rho)$	$5mn + \lceil 2/3(m^2 + n^2) \rceil + 2\lambda$
	Bits	(2,43,38,17,13)	2365	29416	10622

5.3 Performance Comparison

We have implemented both Jain et al. [23] schemes and ours. In both implementations we have used the NTL library from Victor Shoup. The implementations were written in C++ and the benchmarks were conducted on a 2.9 GHz Quad-Core Intel Core i7 with 16GB of LPDDR3 RAM clocked at 2133MHz.

Table 4. Commitment scheme performance comparison.

Commitment scheme			This work		
Jain et al.			This work		
Routine	Subroutine	Time [ms]	Routine	Subroutine	Time [ms]
Setup	Generate matrix A	1.303	Setup	Generate matrix G	0.030
Commitment	Generate random vector r	negl	Commitment	Generate random vector s	negl
	Generate error vector e	0.168		Generate error vector e	1.800
	Compute commitment c	0.029		Compute commitment c	0.025
	Total	0.197		Total	1.825
Verification	Recover error vector e	0.029	Verification	Recover error vector e	0.0250
	Compute hamming weight of e	0.001		Compute rank of e	0.0160
	Total	0.030		Total	0.041

Table 4 depicts the performance in milliseconds of the two commitment schemes. In Table 5, we compare the performance of both Hamming and rank metric variants for the knowledge of a valid opening. Table 6 show the performance of the linear and multiplicative relations. For the latter two modes the

Table 5. Knowledge of Valid Opening performance comparison.

Knowledge of Valid Opening			This work						
Jain et al.			This work						
Routine 0	Subroutine	fTime [ms]	Routine	Subroutine	Time[ms]				
Proof gen.	Generate π	0.552	Proof gen.	Generate $\Pi_{P,Q}$	0.135				
	Generate random vectors	negl		Generate random vectors	negl				
	Comm. 0	t_0		0.032	Comm. 0	r_0	0.020		
		$E(t_\pi, t_0)$		0.400		$E(r_{P,Q}, r_0)$	0.035		
		$Com(E(t_\pi, t_0))$		0.200		$Com(E(r_{P,Q}, r_0))$	1.860		
	Comm. 1	t_1		0.038		Comm. 1	r_1	0.044	
		$E(t_1)$		0.391	$E(r_1)$		0.019		
		$Com(E(t_1))$		0.203	$Com(E(r_1))$		1.809		
	Comm. 2	t_2		0.040	Comm. 2	r_2	0.044		
		$E(t_2)$		0.396		$E(r_2)$	0.018		
		$Com(E(t_2))$		0.197		$Com(E(r_2))$	1.736		
	Total			1.897	Total		5.585		
	Proof ver.	Verif. 0		$Ver(c_0, E(t_\pi, t_0), s_0)$	0.423	Proof ver.	Verif. 0	$Ver(c_0, E(r_{P,Q}, r_0), s_0)$	0.077
				$Ver(c_1, E(t_1), s_1)$	0.426			$Ver(c_1, E(r_1), s_1)$	0.064
				$t_0 + \pi^{-1}(t_1) \in \text{Im}(A)$	170.888			$r_0 + \Pi_{r_0}^{-1}(r_1) \in \text{Im}(G)$	2.559
Verif. 1		$Ver(c_0, E(t_\pi, t_0), s_0)$	0.424	Verif. 1	$Ver(c_0, E(r_{P,Q}, r_0), s_0)$		0.080		
		$Ver(c_2, E(t_2), s_2)$	0.444		$Ver(c_2, E(r_2), s_2)$		0.066		
		$t_0 + \pi^{-1}(t_2) + y \in \text{Im}(A)$	175.526		$r_0 + \Pi_{r_0}^{-1}(r_2) + y \in \text{Im}(G)$		2.47		
Verif. 2		$Ver(c_1, E(t_1), s_1)$	0.459	Verif. 2	$Ver(c_1, E(r_1), s_1)$		0.064		
		$Ver(c_2, E(t_2), s_2)$	0.446		$Ver(c_2, E(r_2), s_2)$		0.064		
		$w_H(t_1 + t_2)$	0.001		$w_R(r_1 + r_2)$		0.018		
Total			349.037	Total			5.462		

comparison is more brief as the subroutines are mostly the same as in Table 5. The key outcomes of this comparison are the following. For the commitment scheme, the generation of the commitment is slower in the rank metric because of the algorithm that generates an error of a given rank. The verification of the commitment is slower in the rank metric because we have to compute the rank of a matrix rather than the Hamming weight of a binary vector. The generation of matrix A is slower than matrix G due to their different size. The generation of the proof of Knowledge of a Valid opening, Linear relations and Multiplicative Relations achieve similar timings for both variants. For the verification of the proofs, the performance of the rank metric is around 100 times better than the Hamming metric. This happens because of the large linear systems that have to be solved in the Hamming case.

Table 6. Linear and multiplicative relations performance comparison.

Linear Relations					
Jain et al.			This work		
Routine	Subroutine	Time [ms]	Routine	Subroutine	Time[ms]
Proof Gen.	Generate permutation P	3.039	Proof Gen.	Generate matrices P and Q	0.869
	Generate random vectors	0.030		Generate random vectors	0.568
	Generate commitments	4.502		Generate commitments	10.539
	Total	7.571		Total	11.976
Proof Ver.	Verification 1	524.682	Proof Ver.	Verification 1	4.780
	Verification 2	525.985		Verification 2	4.765
	Verification 3	2.344		Verification 3	0.512
	Total	1053.011		Total	10.057
Multiplicative Relations					
Jain et al.			This work		
Routine	Subroutine	Time [ms]	Routine	Subroutine	Time[ms]
Proof Gen.	Generate permutation P	72.462	Proof Gen.	Generate matrices P and Q	28.432
	Generate random vectors	0.177		Generate random vectors	0.120
	Generate commitments	16.130		Generate commitments	47.430
	Total	88.769		Total	75.982
Proof Ver.	Verification 1	2634.96	Proof Ver.	Verification 1	22.402
	Verification 2	2580.93		Verification 2	22.760
	Verification 3	6.508		Verification 3	2.463
	Total	5222.398		Total	47.625

6 Conclusion

We showed that quantum resistant zero-knowledge proof protocols can be built upon the Rank Syndrome Decoding problem in an efficient way. In particular, we implemented the building blocks needed for a zero-knowledge protocol to prove the relation among two committed values for any circuit. Our protocol is quasi-linear in the size of the circuit, has a soundness error of $2/3$, and is quantum resistant. We hope this work to be a starting point to build even more efficient zero-knowledge protocols based on the RSD problem.

A Sigma Protocol

We give the definition of Σ -protocol, which is the basis of the protocols we present. This definition might help understanding the security proof in Sect. B.

Definition 4 (Σ -protocol). *Let (P, V) be a two-party protocol, where V is PPT, and let R be a binary relation. Then (P, V) is called a Σ -protocol for R with challenge set C , public input y and private input w , if and only if it satisfies the following conditions:*

- 3-move form: *The protocol is of the following form:*
 - P computes a commitment t and sends it to V .
 - V draws a challenge $c \leftarrow_s C$ and sends it to P .
 - P sends a response s to V .

Depending on the protocol transcript (t, c, s) , V accepts or rejects the proof. The protocol transcript (t, c, s) is called accepting, if V accepts the protocol run.

- Completeness: V accepts whenever $(y, w) \in R$.
- Special soundness: There exists a PPT algorithm E (the knowledge extractor) which takes a set $\{(t, c, s_c) \text{ s.t. } c \in C\}$ of accepting transcripts with the same commitment as inputs, and outputs w' such that $(y, w') \in R$.
- Special honest-verifier zero-knowledge: There exists a PPT algorithm S (the simulator) taking y and $c \in C$ as inputs, and which outputs triples (t, c, s) whose distribution is (computationally) indistinguishable from accepting protocol transcripts generated by real protocol runs.

B Proof of Theorem 3

Proof. We need to prove that the protocol is 3-move, complete, sound and zero-knowledge.

- 3-move: the protocol is 3-move by design.
- Completeness: it is easy to see that the if the protocol is honestly run by a prover, then it always returns true.
 - If $ch = 0$ then $r_0 + \Pi_{r_0}^{-1}(r_1) = v \cdot G + f + \Pi_{P,Q}^{-1}(\Pi_{P,Q}(f)) = v \cdot G \in \text{Im}(G)$ and P, Q are two binary matrices of size $m \times m$ and $n \times n$ respectively.
 - If $ch = 1$ then $r_0 + \Pi_{r_0}^{-1}(r_2) + y = v \cdot G + f + \Pi_{P,Q}^{-1}(\Pi_{P,Q}(f + e)) + x \cdot G + e = (v + x) \cdot G \in \text{Im}(G)$.
 - If $ch = 2$ then $w_R(r_1 + r_2) = w_R(\Pi_{P,Q}(f) + \Pi_{P,Q}(f + e)) = w_R(\Pi_{P,Q}(f + f + e)) = w_R(e) = \rho$.
- Special soundness: we first assume that the values c_0, c_1, c_2 and openings for all challenges $ch \in \{0, 1, 2\}$ have been fixed in such a way that that V accepts on all of them. Since the underlying commitment scheme Com is perfectly binding and the compression function E collision resistant, then the openings to identical commitments have to be identical when different challenges are given, or a collision for E should be found. We have that $\Pi_{P,Q}^{-1}(r_1 + r_2) + y \in \text{Im}(G)$ thanks to the verification equations for $ch = 0$ and $ch = 1$, and thus that $y = x' \cdot G + \Pi_{r_{P,Q}}^{-1}(r_1 + r_2)$, where $x' = (s' || m')$ can be easily computed. Now, a valid witness of (G, y) is given by $(s', m', \Pi_{r_{P,Q}}^{-1}(r_1 + r_2))$, since $w_R(r_1 + r_2) = \rho$. It is important to highlight that the input of the commitment scheme is the result of a collision resistant function, therefore, the probability for the above mentioned equations to not be correct is negligible, as it is the probability of a collision in a collision resistant compression function.
- Honest Verifier Zero-knowledge: we need to prove that there exist an efficient simulator Sim , which, for each challenge $ch \in \{0, 1, 2\}$, outputs an accepting protocol transcript that is computationally indistinguishable from a real protocol transcript performed by an honest prover for the given challenge ch . The simulator can be described as follows:

- $\text{ch} = 0$: Sim computes c_0, c_1 as in the protocol, and c_2 as a commitment to 0. It is straightforward that the distribution of $c_0, c_1, r_{P,Q}, r_0, r_1$ is identical to the one of a real transcript. Furthermore, the fact that the commitment scheme Com is computationally hiding implies that the distribution of c_2 is computationally indistinguishable from the real protocol runs.
- $\text{ch} = 1$: Sim selects uniformly at random the values $Q \leftarrow_s \mathcal{M}_{m,m}^*(\mathbb{F}_q)$, $P \leftarrow_s \mathcal{M}_{n,n}^*(\mathbb{F}_q)$, $b \leftarrow_s (\mathbb{F}_{q^m})^k$, $a \leftarrow_s (\mathbb{F}_{q^m})^n$. Then, computes the commitments $c_0 = \text{Com}(E(P, Q, b \cdot G + y + a))$ and $c_2 = \text{Com}(E(\Pi_{P,Q}(a)))$. The value of c_1 is computed as commitment to 0. The openings of c_0, c_2 are verified correctly by the verifier. The distribution of the openings is correct because of the perfect uniform distribution of r_2 in the real protocol run and $\Pi_{P,Q}(a)$ in the simulated run in $\mathbb{F}_{q^m}^n$, and of the permutations in the set of permutations. Regarding the opening of c_0 , notice that in the real protocol run, it holds $r_{P,Q} = v \cdot G + f$, where v is uniformly at random, and $f = \Pi_{P,Q}^{-1}(r_2 + e)$. In the simulated transcript the content of c_0 is $b \cdot G + y + a = (b + x) \cdot G + (a + e)$. The distributions of c_0 and c_2 and their openings are perfectly simulated, since v and $b + x$ are both uniformly random, and the terms f and $a + e$ are uniquely determined by the contents of c_0 and c_2 . Finally, the distribution of c_1 is computationally indistinguishable by the assumed hiding property of Com.
- $\text{ch} = 2$: Sim selects uniformly at random $a \leftarrow_s (\mathbb{F}_{q^m})^n$, $b \leftarrow_s (\mathbb{F}_{q^m})^n$ such that $w_R(b) = \rho$. It computes c_0 as commitment to 0. $c_1 = \text{Com}(E(a))$ $c_2 = \text{Com}(E(a + b))$. As in the case of $\text{ch} = 1$, the binding property of Com implies that the distributions of c_0 is computationally indistinguishable from real protocol runs. Furthermore, the behavior of an honest prover can be perfectly simulated by c_1 and c_2 and their openings.

□

References

1. Aguilar, C., Blazy, O., Deneuville, J.C., Gaborit, P., Zémor, G.: Efficient encryption from random quasi-cyclic codes. arXiv preprint [arXiv:1612.05572](https://arxiv.org/abs/1612.05572) (2016)
2. Aguilar, C., Gaborit, P., Schrek, J.: A new zero-knowledge code based identification scheme with reduced communication. In: 2011 IEEE Information Theory Workshop (ITW), pp. 648–652. IEEE (2011)
3. Aragon, N., Gaborit, P., Hauteville, A., Tillich, J.P.: Improvement of Generic Attacks on the Rank Syndrome Decoding Problem, October 2017. <https://hal.archives-ouvertes.fr/hal-01618464>, working paper or preprint
4. Bardet, M., et al.: An algebraic attack on rank metric code-based cryptosystems. arXiv preprint [arXiv:1910.00810](https://arxiv.org/abs/1910.00810) (2019)
5. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 390–420. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_28
6. Bellini, E., Caullery, F., Gaborit, P., Manzano, M., Mateu, V.: Improved veron identification and signature schemes in the rank metric. In: 2019 IEEE International Symposium on Information Theory (ISIT), pp. 1872–1876. IEEE (2019). <https://doi.org/10.1109/ISIT.2019.8849585>

7. Bellini, E., Caullery, F., Hasikos, A., Manzano, M., Mateu, V.: Code-based signature schemes from identification protocols in the Rank Metric. In: Camenisch, J., Papadimitratos, P. (eds.) CANS 2018. LNCS, vol. 11124, pp. 277–298. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00434-7_14
8. Ben-Or, M., et al.: Everything provable is provable in zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 37–56. Springer, New York (1990). https://doi.org/10.1007/0-387-34799-2_4
9. Benhamouda, F., Krenn, S., Lyubashevsky, V., Pietrzak, K.: Efficient zero-knowledge proofs for commitments from learning with errors over rings. In: Pernul, G., Ryan, P.Y.A., Weippl, E. (eds.) ESORICS 2015. LNCS, vol. 9326, pp. 305–325. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24174-6_16
10. Berlekamp, E., McEliece, R., Van Tilborg, H.: On the inherent intractability of certain coding problems (corresp.). *IEEE Trans. Inf. Theory* **24**(3), 384–386 (1978)
11. Cayrel, P.L., Lindner, R., Rückert, M., Silva, R.: Improved zero-knowledge identification with lattices. *Tatra Mountains Math. Publ.* **53**(1), 33–63 (2012)
12. Cayrel, P.L., Véron, P., Alaoui, S.M.E.Y.: A zero-knowledge identification scheme based on the q -ary syndrome decoding problem. In: International Workshop on Selected Areas in Cryptography, pp. 171–186 (2010)
13. Chabaud, F., Stern, J.: The cryptographic security of the syndrome decoding problem for rank distance codes. In: Kim, K., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 368–381. Springer, Heidelberg (1996). <https://doi.org/10.1007/BFb0034862>
14. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the Theory and Application of Cryptographic Techniques, pp. 186–194 (1986)
15. Gabidulin, E.M.: Theory of codes with maximum rank distance. *Problemy Peredachi Informatsii* **21**(1), 3–16 (1985)
16. Gaborit, P., Girault, M.: Lightweight code-based identification and signature. In: 2007 IEEE International Symposium on Information Theory, pp. 191–195. IEEE (2007)
17. Gaborit, P., Ruatta, O., Schrek, J.: On the complexity of the rank syndrome decoding problem. *IEEE Trans. Inf. Theory* **62**(2), 1006–1019 (2016)
18. Gaborit, P., Schrek, J., Zémor, G.: Full cryptanalysis of the Chen identification protocol. In: International Workshop on Post-Quantum Cryptography, pp. 35–50 (2011)
19. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM (JACM)* **38**(3), 690–728 (1991)
20. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18**(1), 186–208 (1989)
21. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, pp. 212–219. ACM (1996)
22. Impagliazzo, R., Yung, M.: Direct minimum-knowledge computations (extended abstract). In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 40–51. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_4
23. Jain, A., Krenn, S., Pietrzak, K., Tentes, A.: Commitments and efficient zero-knowledge proofs from learning parity with noise. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 663–680 (2012)

24. Kawachi, A., Tanaka, K., Xagawa, K.: Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 372–389. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_23
25. Lau, T.S.C., Tan, C.H., Prabowo, T.F.: Key recovery attacks on some rank metric code-based signatures. In: Albrecht, M. (ed.) IMACC 2019. LNCS, vol. 11929, pp. 215–235. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35199-1_11
26. Ling, S., Nguyen, K., Stehlé, D., Wang, H.: Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 107–124. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_8
27. Loidreau, P.: Properties of codes in rank metric. arXiv preprint cs/0610057 (2006)
28. Lyubashevsky, V.: Lattice-based identification schemes secure under active attacks. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 162–179. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78440-1_10
29. Martínez, R., Morillo, P.: RLWE-based zero-knowledge proofs for linear and multiplicative relations. In: Albrecht, M. (ed.) IMACC 2019. LNCS, vol. 11929, pp. 252–277. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35199-1_13
30. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 203–228 (2015)
31. Micciancio, D., Vadhan, S.P.: Statistical zero-knowledge proofs with efficient provers: lattice problems and more. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 282–298. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_17
32. Ourivski, A.V., Johansson, T.: New technique for decoding codes in the rank metric and its cryptography applications. *Probl. Inf. Transm.* **38**(3), 237–246 (2002)
33. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**(5), 1484–1509 (1997)
34. Smart, N.: Letter to NIST on standardizing new cryptographic standards (2016). <https://zkp.science/docs/Letter-to-NIST-20160613-Advanced-Crypto.pdf>
35. Stern, J.: A new identification scheme based on syndrome decoding. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 13–21. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_2
36. Stern, J.: A new paradigm for public key identification. *IEEE Trans. Inf. Theory* **42**(6), 1757–1768 (1996)
37. Various: Zero knowledge proof standardization: An open industry/academic initiative. <https://zkproof.org/index.html>. Accessed 20 Jan 2020
38. Véron, P.: Improved identification schemes based on error-correcting codes. *Commun. Comput. Appl. Algebra Eng.* **8**(1), 57–69 (1997)
39. Wachter-Zeh, A.: Decoding of block and convolutional codes in rank metric. Ph.D. thesis, Universität Ulm (2013)
40. Xie, X., Xue, R., Wang, M.: Zero knowledge proofs from ring-LWE. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) CANS 2013. LNCS, vol. 8257, pp. 57–73. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02937-5_4



A Secure Algorithm for Rounded Gaussian Sampling

Séamus Brannigan^(✉), Maire O'Neill, Ayesha Khalid, and Ciara Rafferty

Centre for Secure Information Technologies (CSIT),
Queen's University, Belfast, Northern Ireland
sbrannigan11@qub.ac.uk

Abstract. Presented in this paper is a new Gaussian sampler targeted at lattice-based signatures. It is the first secure algorithm for implementing the *Box-Muller* Gaussian sampling algorithm, which produces continuous Gaussian samples. The samples can be made discrete by rounding and this method has recently been shown to suffice for the purpose of discrete Gaussian sampling for lattice-based signatures. Rounded Gaussians allow quick transformations from samples of low standard deviation to samples with a high standard deviation. This makes them well-suited to producing the wide distributions needed for the target primitives. Current state-of-the-art methods sample wide distributions with multiple samples from a narrow distribution, joined by a convolution algorithm, for each single sample. The number of required samples per output sample grows with the width of the distribution. The rounded Gaussian method allows sampling wide distributions with complexity which is constant with increasing standard deviation. The main contribution of the work is a novel, low-memory algorithm, based on the CORDIC family of algorithms, for the fixed-point and secure evaluation of the elementary functions necessary for the Box-Muller continuous sampling method. It is the first secure, continuous sampler for the production of rounded Gaussian distributions. A proof-of-concept implementation of the algorithm is used to demonstrate that Box-Muller sampling is a competitive alternative to sampling the discrete Gaussian distribution, for lattice-based signatures.

1 Introduction

Over the last couple of decades, the necessity of Post-Quantum Cryptography (PQC) research has been the driving force for investigations into non-traditional algorithms for Public-Key Cryptography (PKC). Popular paradigms can be classified roughly into five areas: code-based [1, 2], hash-based [3], multivariate [4], supersingular isogeny key exchange [5, 6] and lattice-based cryptography (LBC) [7–9]. The latter field is the largest by volume of literature and involves the mathematics of discrete subsets of vector spaces, often drawing on abstract algebra. For a thorough introduction to LBC, see [10].

Cryptography using lattices goes beyond the primitive functionality of key encapsulation by encryption, or key exchange mechanism (KEM), and authentication from digital signatures. LBC has been useful in the pursuit to replace public-key infrastructure (PKI) in a number of ways. For instance, identity-based cryptography (IBC) was shown to be equivalent to PKI based on certificate authorities. Although this is not a new idea, it resurfaced when lattice *trapdoors* were found to be effective in their construction [11] and, importantly, solved problems relating to revocable trust when that trust is based on identity [12]. Similar strides have been made in attribute-based and fully-homomorphic cryptography.

Central to the underlying complexity theory of LBC is the discrete Gaussian distribution over the integers [13]. Sampling from this distribution enables better properties, such as smaller key or signature sizes, for LBC systems. In a setting which otherwise requires only arithmetic between integers within native word lengths, the benefits of Gaussian sampling can only be attained by addressing the complications it brings: higher precision and non-integer arithmetic (including for sampling the integers).

The relevance of the problems addressed in this paper can be seen with the Dilithium scheme [9], which traded Gaussian sampling for uniform random numbers. The scheme has a variant, namely Dilithium-G, which does not make this trade-off and which improves parameters. However, the variant receives less attention because of the difficulty achieving the high standard deviation required. The likely driver of this design decision is a cache attack in [14] on the most highly discussed samplers, namely the cumulative distribution Tables (CDT), Knuth/Yao and Ziggurat samplers. The most punishing finding of this paper: the attack works on samplers which are constant-time (CT), as the memory access is not *wholly sequential* (WS) or *probabilistically uniform* (PU). Another prominent lattice scheme using Gaussian sampling is the Falcon digital signature scheme [15].

The design decisions required to avoid discrete Gaussian sampling DGS are not always favourable, or even possible. It is therefore important for these decisions to be based on the full scope of possible exchanges in time, memory and security. In order to be inherently resistant to cache attacks such as in [14], algorithms need to be WS, or PU, and CT.

Current state-of-the-art Box-Muller implementations rely on calls to the math module of the C standard library, using floating-point arithmetic and being unlikely to ever be WS/CT. Algorithms for which the latter two properties hold have complexities which grow with the standard deviation parameter. Hence, there is a window for improvement in the high σ domain, as the Box-Muller method has a constant complexity in this regard. If the method is worked on to the same extent as others have been, the performance in this domain can be favourable.

1.1 Related Work

It is broadly true that encryption schemes using DGS require only a low value for the standard deviation σ . However, as was shown in [16], in the *New Hope* KEM, a binomial sampler was shown to be sufficient for most applications of low σ . Binomial sampling can be done simply and efficiently and is naturally WS/CT. Where binomial distributions cannot be used, the low standard deviation required poses no problems for a secured, WS/CT, discrete Gaussian sampler. The attention of research remains mostly on digital signatures.

The current state-of-the-art in Gaussian sampling consists of two samplers. The first is the *fast, compact, and constant-time* (FACCT) sampler by Zhao et al. [17]. This sampler achieves the fastest benchmarks for sampling, but with the use of single-instruction multiple-data (SIMD) intrinsics. Without this vectorised acceleration, the algorithm is somewhat slower than the algorithm used for comparison in this paper.

The algorithm compared to in this paper is the Knuth-Yao sampler adapted by Roy in [18]. It is WS/CT and relatively fast, given these constraints. However, the Knuth-Yao sampler is known to have time and space complexities which grow rapidly with standard deviation. The effect is to render Knuth-Yao, on its own, impractical for digital signatures.

The advice, therefore, is to use it as a *base* sampler in Micciancio and Walter's algorithm [19], which uses Gaussian convolution methods to generate samples with large standard deviation from samples with a smaller one. This replaces the quickly growing complexities with slower ones. As the complexity is not constant for these algorithms, the little-researched Box-Muller method offers possible advancements in this area.

This paper builds on the work of Hülsing et al. in [20] who proved that sampling from a continuous Gaussian distribution and rounding the result is as secure as DGS, for lattice-based signature schemes based on rejection sampling. The paper is a theoretical contribution, i.e. it shows that the rounded Gaussian distribution is a secure replacement for the discrete kind in the dense theory of lattices. Although it comes with an implementation of the Box-Muller algorithm, used for continuous Gaussian sampling, this implementation calls the math functions \ln , \cos , \sin and square root of a C++ single instruction, multiple data (SIMD) library using floating point data types.

Moreover, the library used by the paper does not calculate the elementary functions in constant time. In other words, the elementary functions are a black box in the paper, from a security perspective, and the performance metrics of SIMD instructions do not compare fairly to the rest of the literature, as admitted in the paper.

But these and other problems, including the use of floating-point arithmetic (see [21] for why this is an issue), result in the secure evaluation of the Box-Muller functions being an open problem in LBC. The transformative nature of Box-Muller, making it a prime candidate for constant-time sampling, along with the ease at which higher standard deviations are reached compared with discrete sampling, warrant that the method be investigated to the same extent

as the discrete case. This requires that the field of LBC has explicit, portable and secure algorithms in native C using only integers and fixed-point precision, for every function call or subprocess.

1.2 Our Contribution

This paper presents a novel algorithm for sampling rounded Gaussians via the Box-Muller method, including proposals for the WS/CT evaluation of the transcendental functions involved. The method chosen is the CORDIC family of algorithms, which has several attractive qualities for our purposes. For example, the CORDIC *architecture* can be used to calculate all the necessary functions with tuned input parameters and modes of operation. Two of these functions, \cos and \sin , can be evaluated simultaneously with one iteration of CORDIC. Furthermore, the algorithm is sequential in its memory access and algorithmically constant-time - so long as the novel, secure paradigm of operation, outlined in this paper, is adhered to.

We modify the CORDIC algorithm to carry out secure evaluation of the target functions, while maintaining a practical level of efficiency and doing so with a negligibly lightweight memory profile. All logical and comparative procedures use constant-time idioms, no floating-point arithmetic is used and it is expected to be fully secure under all conventional side-channel attacks.

The sampler, being discretised by rounding a continuous distribution, is a more generic approach to reaching higher standard deviations than applying Micciancio and Walters convolution methods, as the steps involved are much simpler and require less overhead. Our sampler has a one-to-one mapping from uniform samples to Gaussian samples, which is true for all values of standard deviation. The performance of the Box-Muller algorithm, therefore, does not incur as large a penalty as that of the Knuth-Yao/Micciancio-Walters approach, on platforms with no hardware-accelerated cryptographic random number generation. The latter approach requires multiple calls to the base sampler, resulting in multiple calls to the cryptographically-secure pseudo-random number generator (CSPRNG).

Large values of the standard deviation parameter are the focus of this paper, but it should also be noted that the Micciancio/Walters construction requires further calls to the base sampler, and more convolution, when the distribution is centered between integers. The complexity grows with the precision of the center. The rounded Gaussian method, of this paper, only requires that the new center be added to each sample. For the most generic distributions, therefore, there is a strong case for investigating the Box-Muller approach.

We compare our algorithm with the Knuth/Yao implementation of [18], in Sect. 4, and show that Box-Muller sampling has competitive performance and memory usage with the ensemble of [18] and [19], for particular application and domain areas. This is done with a proof-of-concept implementation used to demonstrate the potential of the Box-Muller sampler as a competitive alternative to sampling the discrete Gaussian distribution. Such areas where the method is favourable are explored and the next steps for consolidating and expanding its

applicability are identified. In particular, we note that the algorithms of this paper can be optimised further, through adjustments to the algorithm itself and by refining the code of the implementation. The result is a first look at the Box-Muller method for cryptographic use.

2 Method Background

2.1 Box-Muller Gaussian Sampling

The Box-Muller (BM) algorithm [22] is a transformation on two uniform random variables, u_1 and u_2 :

$$v_1 = \sqrt{-2 \ln(u_1)} \cdot \cos(2\pi u_2) \quad (1)$$

and

$$v_2 = \sqrt{-2 \ln(u_1)} \cdot \sin(2\pi u_2). \quad (2)$$

The first, and common, term in these expressions is the inverse of the Gaussian function with unit standard deviation and zero mean. Combined with two sinusoidal random variables π radians out of phase, this *inverse Gaussian* random variable produces a random variable distributed according to a Gaussian distribution.

The Box-Muller algorithm has several advantages. It is lightweight, requiring no memory on top of what is needed for function evaluation and, more importantly for security optimisation, its sampling procedure consists of a transformation from two uniformly distributed samples to two Gaussian samples.

2.2 CORDIC Algorithm for Evaluation of Transcendental Functions

CORDIC is shorthand for *coordinate rotation digital computer*. It is a family of algorithms designed for hardware implementations of the types of function evaluation we are interested in, typically to be used in calculators. It is a serial, iterative method and the number of iterations depends only on the precision. This paper follows the algorithms described in [23].

The functions \cos and \sin can be evaluated with one round of CORDIC in *rotation* mode, described below, under the Euclidean notion of rotation. Natural log can be evaluated with one round of CORDIC in *vectoring* mode under hyperbolic rotation. The square root requires a further round of the latter regime.

The algorithms make transformations to the x and y coordinates of the input vector, using only additions or subtractions and bit shifts. This does not amount to an actual rotation, under either geometry, as the radius does not transform accordingly. The transformations, often referred to as *pseudo-rotations*, are

$$x_{i+1} = x_i + my_i \delta_i \quad (3)$$

and

$$y_{i+1} = y_i - x_i \delta_i. \tag{4}$$

Here, m is 1 if the geometry is Euclidean, referred to as *circular* mode. For *hyperbolic* mode, $m = -1$. The variable δ_i is chosen to be equal to 2^{-i} , in order to avoid multiplication on its early hardware targets. The x_0 , y_0 and θ_0 are all chosen to produce correct output values. The two (non-geometric) modes of execution used in this paper are the *vectoring* and *rotation* modes. They can be composed with the two geometric modes, hyperbolic and circular.

Rotation mode involves beginning on the x -axis ($y_0 = 0$) and rotating towards a final vector. The angle θ_0 is initialised so that rotations towards the final vector brings the θ iteration variable to a final value negligibly close to 0. θ usually represents the input value to the function of interest in vectoring mode.

In vectoring mode, the initial vector is not on the x -axis (but in quadrants 1 or 4) and is rotated towards it. Hence, y is the variable which, when brought to zero, determines the end of the iterations, in this mode. The input to the function is usually parameterised between x_0 and y_0 .

The angle, θ , and radius, R , vary along with the transformations according to

$$\theta_{i+1} = \theta_i - \alpha_i, \tag{5}$$

where

$$\alpha_i = \begin{cases} \tan^{-1}(2^{-i}) & \text{circular/rotation mode} \\ -\tanh^{-1}(2^{-i}) & \text{hyperbolic/vectoring mode} \end{cases} \tag{6}$$

and

$$R_{i+1} = R_i \cdot K_i. \tag{7}$$

Here, K_i is the factor by which each rotation step is out from being a true radius-preserving rotation. Hence,

$$K_i = \sqrt{1 + m \cdot \delta_i^2} \tag{8}$$

and, because this only depends on i , the refactoring of the radius can be done in one multiplication, if needed. The resultant factor is

$$K_n = \prod_{i=0}^{n-1} K_i. \tag{9}$$

The rotations become increasingly smaller and are always towards the final vector. So, when the i^{th} vector takes a step in the direction of \mathbf{v} and passes it, the $(i + 1)^{\text{th}}$ rotates in the opposite direction and converges to the final vector \mathbf{v} . This involves adding where previously there was subtraction, and vice versa. Hence, the procedure for adjusting the direction of rotation must be made constant-time, as addition and subtraction are not the same operation on computer hardware.

The algorithm, in general, transforms input vectors $(x_0, y_0, \theta_0)^T$ to output vectors $(x_n, y_n, \theta_n)^T$. There are two initialisations of CORDIC of importance in

this paper. The first, namely *circular/rotation* mode, approximates the sin and cos functions to n bits of precision, using n rotations as described above. The initial and final conditions are shown by the vector transformation in Eq. (11).

$$\begin{pmatrix} 1/K_n \\ 0 \\ \theta \end{pmatrix} \xrightarrow[\text{circ/rot}]{\text{CORDIC}} \begin{pmatrix} \cos \theta \\ \sin \theta \\ 0 \end{pmatrix}, \tag{10}$$

The second set of initial conditions, for *hyperbolic/vectoring* mode, allows the calculation of both the square root and natural log functions. However, we must parameterise the intended input to this function (let this be w) differently for each function, to obtain initial vectors with which the correct final vectors are computed. The general procedure is shown in the vector transformation in Eq. (11), but to calculate \sqrt{w} , let $x_0 = w + \frac{1}{4}$ and $y_0 = w - \frac{1}{4}$, and to calculate $\ln w$, let $x_0 = w + 1$ and $y_0 = w - 1$. For the latter, we need the identity $\ln w = 2 \tanh^{-1}(y/x)$.

$$\begin{pmatrix} x_0 \\ y_0 \\ 0 \end{pmatrix} \xrightarrow[\text{hyp/vect}]{\text{CORDIC}} \begin{pmatrix} \sqrt{x_0^2 - y_0^2} \\ 0 \\ \tanh^{-1}(y_0/x_0) \end{pmatrix} \tag{11}$$

While there may exist a set of conditions for hyperbolic/vectoring mode which calculates the composite function $\sqrt{-2 \ln x}$, the parameterisation means the two functions must be calculated sequentially in this setting.

The domains of convergence for these algorithms are limited and the following identities are used to expand the accessible domain to meet the needs of high precision Box-Muller sampling.

To calculate the two trigonometric functions for angles between 0 and 2π , as is needed for Box-Muller, we use the CORDIC algorithm on a reduced angle between 0 and $\frac{\pi}{2}$ according to

$$\sin \left(Q \frac{\pi}{2} + D \right) = \begin{cases} \sin D & \text{if } Q \bmod 4 = 0 \\ \cos D & \text{if } Q \bmod 4 = 1 \\ -\sin D & \text{if } Q \bmod 4 = 2 \\ -\cos D & \text{if } Q \bmod 4 = 3 \end{cases} \quad |D| < \frac{\pi}{2} \tag{12}$$

and

$$\cos \left(Q \frac{\pi}{2} + D \right) = \begin{cases} \cos D & \text{if } Q \bmod 4 = 0 \\ \sin D & \text{if } Q \bmod 4 = 1 \\ -\cos D & \text{if } Q \bmod 4 = 2 \\ -\sin D & \text{if } Q \bmod 4 = 3 \end{cases} \quad |D| < \frac{\pi}{2}. \tag{13}$$

Similarly, the range reduction for the hyperbolic tangent and square root functions proceed according to

$$\tanh^{-1} (1 - M2^E) = \tanh^{-1} (T) + (E/2) \ln 2 \quad 0.17 < T < 0.75, \tag{14}$$

where $T = (2 - M - M2^{-E}) / (2 + M - M2^{-E})$ for $0.5 \leq M < 1$ and $E \geq 1$, and

$$\sqrt{M2^E} = \begin{cases} 2^{\frac{E}{2}} \sqrt{M} & \text{if } E \bmod 2 = 0 \\ 2^{\frac{E+1}{2}} \sqrt{M/2} & \text{if } E \bmod 2 = 1 \end{cases} \quad \begin{cases} 0.5 \leq M < 1 \\ 0.25 \leq M/2 < 0.5 \end{cases} \quad (15)$$

3 Secure CORDIC Algorithm

This section details the proposed secure version of the CORDIC algorithm in C-style pseudocode, hence, for example the \wedge operator is the xor, as opposed to the logical and operation. The algorithm is capable of using arbitrarily high-precision fixed-point data types, in multiples of 64 bits. Unsigned 64-bit integer arrays are used for this fixed-point representation. The variable *size* denotes the number of such integers contained in the array, which depends on the precision to be used. The fixed point representation is little endian. Where possible, 32-bit unsigned integers are used for carrying out operations and some of the functions used have 32-bit, 64-bit and 64-bit-array versions.

3.1 Fundamental Arithmetic

The algorithm and implementation adheres to the WS/CT property described in Sect. 1. Therefore, all logic must be carried out by changing values of the same sequential memory addresses, as opposed to selecting memory addresses containing different values. Hence, constructs such as *if/else* or the ternary operator must be replaced with such sequential functions.

The Algorithm described in Sect. 2.2 can be viewed as a series of additions and subtractions on the three variables x, y and θ . In each mode combination used in this paper, one of these variables is driven to zero and will cross it repeatedly, becoming closer on each iteration. For circular/rotation mode, that variable is θ , which keeps track of the angle between the initial and final vectors, and for hyperbolic/vectoring mode it is y which is driven towards zero which is parameterised on the input.

The variable crossing zero must be subtracted from, if it is greater than zero, and added to, if it is less than zero. As this needs to be done wholly sequentially, so that memory access does not depend on input, and in constant time, the main challenge is synthesising a procedure which can add and subtract with a single add *or* subtract operation.

The approach used in this paper relies on the fact that, for example, subtracting one fixed-point number from another can be done by adding the *signed* representation of that number, i.e. the *two's complement*. The same can be done for addition with a subtract operator. By allowing the variable to overflow and underflow in this way, but keeping track of the sign in a separate variable, the constant-time *change in direction* can be implemented.

In what follows, the `ct_` prefix indicates *constant-time*, although all functions are. The prefix `fp_` indicates a function on a multi-precision fixed-point data type.

The letter u followed by a 64 or 32 denotes an unsigned integer data type with that many bits.

Here, we list some of the less verbose, and more fundamental, algorithms used in this work.

- `ct_lt_uX(x, y)` returns $x < y$ where X is the number of bits, 64 or 32, of x and y . Evaluates $(x \wedge ((x \wedge y) | ((x - y) \wedge y))) \ggg X - 1$.
- `ct_gt_uX(x, y)` returns $x > y$ by evaluating `ct_lt_uX(y, x)`.
- `ct_select(c, a, b, bit)` evaluates $c = \text{bit} ? a : b$, where a , b and c can be fixed-point arrays or unsigned native integers, by $c[i] = (\text{mask} \& (a[i] \wedge b[i])) \wedge b[i]$, for each element in the array or once for native integers. The mask variable is initialised as $-\text{bit}$.
- `overflow(r, a, b)` returns the overflow when the operands a and b are added to give r , by evaluating `ct_lt_uX(r, ct_select_uX(a, b, ct_lt_uX(a, b)))`.
- `underflow(r, a)` returns the underflow of $r = a - b$ by evaluating `ct_gt_uX(r, a)`.
- `fp_sub(c, a, b, size)` subtracts u64 arrays a and b to give c by initialising `borrow = 0`; and evaluating $t = a[i] - b[i]$; `tborrow = underflow(t, a[i])`; $c[i] = t - \text{borrow}$; `borrow = tborrow | underflow(c[i], t)`; for all elements in the arrays. It then returns `borrow`.
- `fp_add(c, a, b, size)` carries out a similar routine to `fp_sub` using overflow instead of underflow and returns the 1 or 0 to carry.
- `fp_lsh_by_one(out, x, size)` essentially doubles x and writes it to the variable `out`. It initialises `carry=0` and computes $\text{out}[i] = (x[i] \ll 1) + \text{carry}$; $\text{carry} = (x[i] \& 1 \lll 63) \ggg 63$; for each array element.
- `fp_mul(c, a, b, size_a, size_b)` multiplies a and b , but can take different sizes for each input.
- `cond_inc(x, cond, size)` conditionally increments x and overwrites it.

Algorithm 1: `fp_rsh_non_word(out, x, a, w, size)`

Input : x : u64 array to be shifted
 a : amount to shift by mod 64,
 w : # of whole u64s to shift by
size: Number of u64 blocks in arrays

Output: `out = x >> (64 · l + a)`

```

1 carry = 0
2 i = w
3 while (i < size - 1) do
4     carry = x[i + 1] <<< (64 - amt)
5     out[i - w] = (x[i] >>> amt) + carry
6     i ++
7 out[i - l] = x[size - 1] >>> amt
8 for ( i = size - w; i < size; i ++ ) do
9     out[i] = 0

```

Also required is a WS/CT right shift function, `fp_rsh()`, which is *not* constant-time because its use in the algorithms throughout do not require it to be. The function calls another, named `fp_rsh_word()`, when the number of bits to shift by is a multiple of 64. This is done by copying u64 words down the array. If the number of bits is not a multiple of 64, it calls the `fp_rsh_non_word()` function. Only the latter function is shown here in Algorithm 1. The `fp_rsh()` function takes the same arguments as this and the `fp_rsh_word()` function drops the argument a . All uses of the right-shift function are sequential in nature (the same values of a and w are input at the same stage of each iteration) and the different paths do not depend on secret data.

These functions form the basis of the algorithms for evaluating all the Box-Muller functions. The algorithms described thus far are all very well suited to applications in hardware, as well as the application here: constant-time, sequential compilation in software. The methods described in Sects. 3.3 and 3.4, which follow, are all well-suited to use cases where hardware or resource is expensive, requiring no multiplication and low code and memory space.

3.2 Constant-Time, Sequential Rotations

Algorithm 2 shows how the x , y and θ arrays are adjusted in the main procedure for the sequential and constant-time evaluation of the CORDIC algorithm. Given a boolean value *change*, determined by the θ variable in circular/rotation (C/R) mode and the y variable in hyperbolic/vectoring (H/V) mode, the *adjust_rotation* function ensures that the values to be added or subtracted are converted to their two's complement, or not. This drives the rotation in constant time. We declare here, without detail, two functions similar to `adjust_rotation`, but which either allow two fixed-point variables to be adjusted independently of each other, or allow one such variable to be adjusted. These are the `xy_adjust(x, y, Sx, Sy, size)` and `y_adjust(y, Sy, size)` functions, with x and y , the fixed-point numbers to be made conditionally two's-complement, of type u64[] and S_x and S_y , the *sign* variables, encoding those conditions.

Algorithm 2: `adjust_rotation(x, y, θ , change, size)`

Input : x, y, θ : u64 arrays

change : bool

size: Number of u64 blocks in arrays

Output: x, y, θ : rotated coordinates

```

1 xmask = -change
2 ymask = -change
3  $\theta$ mask = -change
4 for ( i = 0; i < size; i++ ) do
5     x[i] = x[i]  $\wedge$  xmask
6     y[i] = y[i]  $\wedge$  ymask
7      $\theta$ [i] =  $\theta$ [i]  $\wedge$   $\theta$ mask
8 cond_inc(x, change, size)
9 cond_inc(y, change, size)
10 cond_inc( $\theta$ , change, size)
```

The convention used to keep track of this sign variable is to let it be represented with a u32 data type. Then, the state of being below zero can be assigned to the maximum possible value of this data type and the state of being above is assigned to zero. It is required, for the cos and sin functions, to allow S_θ to be 1, representing that the θ value is positive, but also ≥ 1 . This is an edge case, being only permissible at the beginning and for certain inputs, those close to the maximum $\frac{\pi}{2}$ (after reduction to first quadrant).

The algorithm must detect edge cases and deal with them in constant time. These edge cases are slightly different for the two modes used. In C/R mode, the x and y variables sometimes fall below zero, despite the initial and average conditions being in the first quadrant. When this happens, the right shift cannot occur without first using `xy_adjust` before and after it.

In H/V mode, the y variable crosses zero by design, but the x remains positive. The former, hence, uses `y_adjust` while the latter uses the `x_shift_int` in Algorithm 3, which deals with the x value now being able to cross 1 and the possible integer value which is not accounted for in the x shift in Algorithm 5. Each mode has its own constraints based around the values denoted as *change*, S_x , S_y and S_θ , which determine the sign and/or the possible integer part of a value.

Algorithm 3: `x_shift_int`(x , cond, i , j , size)

Input : x : u64 arrays
 cond: value of integer
 i, j : indexes in main algorithm loop
 size: Number of u64 blocks in arrays
Output: x : Accounts for missing integer in shift
1 return $x[\text{size}-1-i] += 1 \ll (63 - (j - 1 - \text{cond}))$

3.3 Reduced Cos and Sin Algorithm

The edge conditions for the C/R mode occur when the rotated vector passes either from quadrant 1 to 2, or from quadrant 1 to 4, in which cases the x and y , respectively, fall below zero. The logic tables detailing the possible situations are shown in Table 1. The possible sign values, e.g. S_x , do not map in an obvious way to the value which tells us whether the operation on, say, x overflowed. We denote the overflow variable with a subscript.

The algorithm for calculating cos and sin between 0 and $\pi/2$ is as shown in Algorithm 4. The table containing the value of $\tan^{-1}(2^{-i})$ in the i^{th} index is called `cordic_circ_table`. The x value is initialised to the inverse of K_n , described in Eqs. (8) and (9), and y is initialised to zero. As input, a number is generated uniformly between zero and $\frac{\pi}{2}$, while the fixed-point, fractional component becomes θ and the integer component becomes S_θ .

Determining the logic that controls the signs of x , y and θ is not a straightforward process. It is more easily ascertained through the logic tables and results in logic which is more complicated for C/R mode than for H/V mode. The *change* condition to determine the direction of rotation is, in the C/R case, 1, if the sign of θ is the value obtained by the subtraction $0 - 1$, and 0, otherwise. A variable

Table 1. The allowed states for the variables by which the signs of x (left table) and y (right table) are controlled.

x_{overflow}	change	$S_{x_{i-1}}$	S_{x_i}	y_{overflow}	change	$S_{y_{i-1}}$	S_{y_i}
1/0	1/0	0	0	1/0	1/0	0	0
1	0	0	1	0	1	0	1
0	1	1	0	1	0	1	0
1	1	1	1	0	0	1	1

named gtz is set up to control when the amount to be subtracted from S_θ goes to zero. This way, the value to be subtracted can synthesise addition from the subtraction operation, in constant time, and S_θ can switch between the values -1 (unsigned, $\theta < 0$), 0 ($\theta \geq 0$) and 1 ($1 \leq \theta < \frac{\pi}{2}$).

3.4 Reduced Natural Logarithm and Square Root Functions

The algorithms in this section work similarly to those in Sect. 3.3. The core differences are that y is now the variable driven to zero and the add and subtract operations on the x and y coordinates are now aligned, whereas previously they were anti-aligned. We do not get the advantage of being able to calculate each function concurrently, as the result of one is input to the other, however there are several favourable factors which help to limit the overhead caused by this, as explained.

The square root function relies on keeping the transformed x coordinate. Thus, the transformations can be done without heed of the angle θ and all of its operations. This function also benefits from converging in half the number of iterations required for the others. Although the natural logarithm does not have these advantageous properties (e.g. it is the transformation of θ and requires all 3 variables), both functions in this section have simpler, and more performant, logic than those in the previous section. We wish to find $\ln(w)$ where $0.5 \leq w < 1$. Initialisation is parameterised in this case, as w is transformed into both x and y . Algorithm 5 shows how this function is evaluated. The square root is the same algorithm as Algorithm 5, except for the initialisations of x and y , which add and subtract the fixed-point representation of one quarter, respectively. Only the x may overflow, but, if it does, it is a definite change of value, not a synthesised operation utilising the overflow. Hence, it is easily dealt with. The inner *for loop* runs only up to and including 32, for the square root function.

3.5 Range Restoration and Full Sampler

To summarise, in Sects. 3.1 to 3.4 it was demonstrated how the Box-Muller functions can be calculated by the CORDIC algorithm in constant time with security constraints. The latter part of Sects. 2.2 outlined the identities to be used in range reduction and restoration, whereby the input is mapped to a (convergent) subset and the output mapped back to the full, desired set. The task is now to implement constant-time transformations around Eqs. (12) to (15).

Algorithm 4: `fp_cos_sin`($x, y, \theta, S_\theta, \text{size}$)

Input : x, y, θ : fixed-point u64 arrays
 $x = 1/K_{\text{circ}}, y = 0, u \stackrel{\$}{\leftarrow} U_{(0, \frac{\pi}{2})}^\lambda$
 $S_\theta = \lfloor u \rfloor$: u32 theta sign counter
 $\theta = u - S_\theta$: u64[]
size: Number of u64 blocks in arrays

Output: $x = \cos(\theta), y = \sin(\theta)$: fixed-point u64 arrays

```

1 for ( i = 0; i < size; i++ ) do
2   for ( j = 0; j < 64; j++ ) do
3     k = i · 63 + j
4      $\theta_{\text{buf}} = \text{cordic\_circ\_table}[k]$ 
5      $x_{\text{buf}} = x$ 
6      $y_{\text{buf}} = y$ 
7     xy_adjust( $x_{\text{buf}}, y_{\text{buf}}, S_x, S_y, \text{size}$ )
8     fp_rsh( $x_{\text{buf}}, x_{\text{buf}}, j, i, \text{size}$ )
9     fp_rsh( $y_{\text{buf}}, y_{\text{buf}}, j, i, \text{size}$ )
10    xy_adjust( $x_{\text{buf}}, y_{\text{buf}}, S_x, S_y, \text{size}$ )
11    change = ct_eq_u32( $S_\theta, 0\text{xfffffff}$ )
12    adjust_rotation( $x_{\text{buf}}, y_{\text{buf}}, \theta_{\text{buf}}, \text{change}, \text{size}$ )
13     $x_{\text{overflow}} = \text{fp\_sub}(x, x, y_{\text{buf}}, \text{size})$ 
14     $y_{\text{overflow}} = S_x$ 
15     $S_x = (\text{ct\_eq\_u32}(\text{change}, x_{\text{overflow}}) \& S_x) |$ 
      (( $x_{\text{overflow}} \wedge \text{change}$ ) &  $\sim S_x$ ) & ( $\sim S_y$  & 1)
16     $x_{\text{overflow}} = y_{\text{overflow}}$ 
17     $y_{\text{overflow}} = \text{fp\_add}(y, y, x_{\text{buf}}, \text{size})$ 
18     $S_y = (\text{ct\_eq\_u32}(\text{change}, y_{\text{overflow}}) \& S_y) |$ 
      (( $y_{\text{overflow}} \wedge \text{change}$ ) &  $\sim S_y$ ) & ( $\sim x_{\text{overflow}}$  & 1)
19     $\theta_{\text{overflow}} = \text{fp\_sub}(x_{\text{buf}}, \theta, \theta_{\text{buf}}, \text{size})$ 
20    fp\_copy( $\theta, x_{\text{buf}}, \text{size}$ )
21     $\theta'_{\text{overflow}} = \sim \theta_{\text{overflow}} \& 1$ 
22    gtz = ( $\theta_{\text{overflow}} \& 1$ ) & (( $S_\theta \& (1 \ll 31)$ ) >> 31)
23    gtz | = ( $\theta'_{\text{overflow}} \& 1$ ) &  $\sim (S_\theta \& 1)$ 
24     $\theta_{\text{overflow}} = \theta_{\text{overflow}} - \theta'_{\text{overflow}}$ 
25     $\theta_{\text{overflow}} = \text{ct\_select\_u32}(0, \theta_{\text{overflow}}, \text{gtz})$ 
26     $\theta_{\text{overflow}} =$ 
      ct\_select\_u32(1,  $\theta_{\text{overflow}}, \text{ct\_eq\_u32}(S_\theta, 1)$ )
27     $S_\theta -= \theta_{\text{overflow}}$ 

```

Algorithm 5: `fp_ln(ln, w, size)`

```

Input :  $w$ : u64[]
          size: Number of u64 blocks in arrays
Output:  $\ln$ : u64[]
1  $x = w$ 
2  $y = w$ 
3  $\theta = 0$ 
4  $\text{change} = 1$ 
5  $S_x = 0$ 
6  $S_y = 1$ 
7 for ( $i = 0$ ;  $i < \text{size}$ ;  $i++$ ) do
8   for ( $j = 1$ ;  $j \leq 64$ ;  $j++$ ) do
9      $k = i * 63 + j$ 
10     $x_{\text{buf}} = x$ 
11     $y_{\text{buf}} = y$ 
12     $\theta_{\text{buf}} = \text{cordic\_hyp\_table}[k]$ 
13     $y_{\text{adjust}}(y_{\text{buf}}, S_y, \text{size})$ 
14     $\text{fp\_rsh}(x_{\text{buf}}, x_{\text{buf}}, j, i, \text{size})$ 
15     $\text{fp\_rsh}(y_{\text{buf}}, y_{\text{buf}}, j, i, \text{size})$ 
16     $y_{\text{adjust}}(y_{\text{buf}}, S_y, \text{size})$ 
17     $x_{\text{shift\_int}}(x_{\text{buf}}, S_x, i, j, \text{size})$ 
18     $\text{adjust\_rotation}(x_{\text{buf}}, y_{\text{buf}}, \theta_{\text{buf}}, \text{change}, \text{size})$ 
19     $x_{\text{overflow}} = \text{fp\_sub}(x, x, y_{\text{buf}}, \text{size})$ 
20     $y_{\text{overflow}} = \text{fp\_sub}(y, y, x_{\text{buf}}, \text{size})$ 
21     $\theta_{\text{overflow}} = \text{fp\_add}(\theta, \theta, \theta_{\text{buf}}, \text{size})$ 
22     $\text{change} = y_{\text{overflow}}$ 
23     $y_{\text{sign}} = y_{\text{overflow}}$ 

```

As a reminder, the main CORDIC algorithms operate on the domains $(0, \frac{\pi}{2}]$ in C/R mode, $(0.5, 1]$ for natural log and square root, the latter only if the input is even, and for odd square root input, $(0.25, 0.75]$. Recall also Eq. (1) and Eq. (2), the equations for the Box-Muller transformation. Expressing these succinctly and letting $g(x) = -2 \ln(x)$, $h(x) = \sqrt{x}$, $X(x) = \cos(x)$, $Y(x) = \sin(x)$ and $f(x) = h(g(x))$, we have

$$v_1 = f(u_1)X(u_2) \quad (16)$$

and

$$v_2 = f(u_1)Y(u_2). \quad (17)$$

Algorithm 6 describes the full algorithm for the secure evaluation of cos and sin. The random fixed-point value $0 < u_1 \leq 1$ is reduced to a value between 0 and $\frac{1}{4}$, as described in Algorithm 8, and multiplied by 2π , using a fixed-point algorithm similar to the addition and subtraction, before being input to the main CORDIC algorithm of Algorithm 4. The quadrant Q is obtained by the reduction and used to restore the output value to the original range, described in Algorithm 11. In Algorithm 7, the exponent E is obtained from the reduction of u_2 and used to restore the output of the main CORDIC for natural log. The reduction and restoration algorithms for natural log can be seen in Algorithms 9 and 12. Before restoration, the value is effectively multiplied by minus two in

lines Lines 3 and 4, using the overflow and *left-shift-by-one* functions, the former being the single-variable version of the *adjust rotation* function of Algorithm 2. The reduction and restoration of square root are slightly different and involve Algorithms 10 and 13.

Algorithm 6:

```

cordic_xy(x, y, u2 size)
  Input : u1: u64 [size]
  Output: x, y: u64 [size]
           Sx, Sy: u32
1 Q = cordic_reduction_circ (
    u2, size)
2 fp_mul(wred, u2, 2π, size, size+1)
3 Sθ = ⌊wred⌋
4 fp_cos_sin (
    x, y, wred + size, Sθ, size)
5 return
   cordic_restoration_circ (
     x, y, Q, size)

```

Algorithm 7: cordic_f(f, u₁, size)

```

Input : u1: u64 [size]
Output: f: u64 [size]
1 E = cordic_reduction_ln(u1, size)
2 fp_ln(f, u1, size)
3 y_adjust(f, 1, size)
4 fp_lsh_one(u1, f, size)
5 wz = cordic_restoration_ln (
    f, E, size)
6 E = cordic_reduction_sqrt (
    f, wz, size)
7 fp_sqrt(f, u1, size)
8 cordic_restoration_sqrt (
    f, E, size)

```

The circular reduction of Algorithm 8 iterates through the quadrants and determines which one the value lies in and the amount by which it overflows. The reduction algorithms for natural log and square root require Algorithms 14 and 15, which calculate the mantissa and exponent in constant-time.

Algorithm 8:

```

cordic_reduction_circ (
w, size)
  Input : w: u64 [size]
  Output: w: u64 [size]
           quadrant: u32
1 quadzero = 0
2 for ( i = 0; i < 4; i++ ) do
3   quadrant +=
   ct_gt_u64(w[size-1], quadzero)
4   quadzero +=  $\frac{1}{4}$ 
5 fp_sub(w, w, =  $\frac{1}{4}$ *quadrant, size)
6 return quadrant

```

Algorithm 9:

```

cordic_reduction_ln (
w, size)
  Input : w: u64 [size]
  Output: w: u64 [size]
           E: u32
1 E = count_leading_zeroes (
    w, size)
2 calc_mantissa(w, E, size)
3 return E

```

The cos and sin are restored by swapping pointers using the masking *select* function and constant-time logic. Line 3 and Line 4 encapsulate the logic of swapping the functions and determining the signs, which are stored in the lowest two bits of a u64.

Algorithm 10: `sqrt_reduction(w, wZ, size)`

```

Input : w: u64 [size]
          wZ: u64, integer component
Output: w: u64 [size]
          E: u32
1 lz = count_leading_zeroes(&wZ, 1)
2 E = 64 - lz
3 fp_rsh(w, w, E, size)
4 w[size-1] += wZ << lz
5 E' = E
6 E += (E&1)
7 E >>= 1
8 fp_rsh(buf, w, 1, size)
9 ct_select(w, buf, w, size, E'&1)
10 return E

```

Algorithm 11: `cordic_restoration_circ(x, y, Q, size)`

```

Input : x y: u64**
          Q: u32
Output: x y: u64 **
          Sx Sy: u32
1 xtempptr = *x
2 ytempptr = *y
3 swap_funcs = Q&1
4 ret = (Q&2) >> 1
5 *x = (u64 *)ct_select_u64((u64)ytempptr,
   (u64)xtempptr, swap_funcs)
6 *y = (u64 *)ct_select_u64((u64)xtempptr,
   (u64)ytempptr, swap_funcs)
7 return ret

```

Algorithm 12:
`cordic_restoration_ln (`
g, E, size)

```

Input : g: u64 [size]
          E: u32
Output: g: u64 [size]
          gZ: u32
1 fp_mul (
   buf, ln2, (u64 [1])E, size, 1)
2 return buf[size] +
   fp_add(g, buf, g, size)

```

Algorithm 13:
`cordic_restoration_sqrt (`
h, E, size)

```

Input : h: u64 [size]
          E: u32
Output: h: u64 [size]
          hZ: u32
1 emod2 = E&1
2 E += emod2
3 esh = E >> 2
4 E = ct_eq_u32(esh, emod2)
5 return calc_sqrt_mantissa (
   h, E, size)

```


Algorithm 14:

```

count_leading_zeroes(x, size)
  Input  : x: u64 [size]
  Output: count: u32
1  n =size *64
2  rsh = n >> 1
3  while rsh != 1 do
4    words = rsh >> 6
5    amt = rsh - words*64
6    fp_rsh(y, x, amt, words, size)
7    inz = ct_isnonzero(y, size)
8    sfn = ct_select_u32 (
9      rsh, 0, inz)
9    n -= sfn
10   ct_select(x, y, x, size, inz)
11   rsh >>= 1
12  fp_rsh(y, x, 1, 0, size)
13  return ct_select_u64(n - 2,
      n - x[0], ct_isnonzero(y, size))

```

Algorithm 15: calc_mantissa(M, E, size)

```

  Input  : M: u64 [size]
          E: u32
  Output: M: u64 [size]
1  l = 0
2  for i = 0; i < size; i ++ do
3    l+ =ct_gt_u32(E, (i + 1) * 64)
4    a = E - 64 * l
5    for i = 0; i < size; i ++ do
6      tomul[i] =
          (u64) ct_eq_u32(i, l) <<< a
7  fp_mul(buf, M, tomul, size, size)
8  fp_copy(M, buf+size, size)

```

Algorithm 16: box_muller(size)

```

  Input  : size: u32: size = λ/64
          (precision λ)
  Output: samples: s32[2]
1  u1  $\overset{\$}{\leftarrow} \mathcal{U}_{(0,1]}^\lambda$ 
2  signs = fp_xy(x, y, u+size, size)
3  fp_f(f, u, size)
4  fix_mul(buf, f, x, size+1, size)
5  samples[0] = (s32) buf[2*size]
6  fix_mul(buf, f, y, size+1, size)
7  samples[1] = (s32) buf[2*size]
8  samples[0] -= signs * (samples[0] <<< 1)
9  samples[1] -= signs * (samples[1] <<< 1);

```

The restoration algorithms for natural log and square root in Algorithms 12 and 13 are further straightforward applications of the techniques throughout this paper. The only thing to note is that `calc_sqrt_mantissa` is the same as `calc_mantissa` except that it returns an integer component. Finally, with these functions we can produce two Gaussian samples in constant-time, using Algorithm 16.

4 Results and Discussion

The timing results for the CORDIC implementation of the Box-Muller algorithm and the Knuth-Yao/Micciancio-Walter ensemble are given in Table 2. These results indicate an upper limit for any future CORDIC-based Box-Muller sampler for cryptographic purposes. Several methods exist for optimising CORDIC, including reducing the number of iterations by a half and introducing modes of

operation better suited to parallelism. For a review of these optimisations, see [24].

The Knuth/Yao-Micciancio/Walter method has an added disadvantage when it comes to sampling from distributions which are centered between integers. It requires further calls to the base sampler, the number of which grows with the precision of the center, and these need to be convoluted in the same way as for arbitrary standard deviation. Hence, for the most arbitrary distributions, a doubling of the time given in Table 2 could be seen for Knuth-Yao/Micciancio-Walter sampling. Not only does our sampler reach arbitrary center and standard deviation with no overhead, compared with sampling a standard deviation of 1 and center 0, but we can also expect to see a halving of the timing results, if the methods of [24] are applied.

Table 2. The time, in seconds, for 1 million samples at 64-bit precision and the code and data memory usage in bytes. Measurements taken on Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz.

Sampler	Time for 10^6 Samples (s)	Overall Memory use (B)
Knuth-Yao (Roy et al.)		
$\sigma = 6.15543$	0.023	
$256 < \sigma < 65536$	0.74	70752
$\sigma > 65536$	1.48	
Box-Muller (CORDIC)		
$\sigma = 1$		
$256 < \sigma < 65536$	3.60	72624
$\sigma > 65536$		

Further to the capacity for optimisation is the ability to implement CORDIC in a parallelised fashion. As Knuth-Yao is a tree traversal algorithm, it does not have the capacity for parallelism within the base sampling method. Any parallelism which can be brought about by calculating the base samples simultaneously can be done with the CORDIC as well, except that the resulting samples for CORDIC will be the generated values required for the cryptographic scheme. In the Knuth-Yao sampler, these will amount to just one sample.

The future directions discussed here, whilst promising, are a small subset of the available methods for making the secure Box-Muller faster. As we require the calculation of 3 or 4 functions, depending on whether f from Sect. 3 is a composite or two separate functions, each can be calculated with different methods than CORDIC (for example, with polynomials) and the optimal combination of methods chosen.

One final advantage of the rounded Box-Muller regime is the ratio of CSPRNG calls to output samples. It is not always the case that a target platform has the availability of a hardware-accelerated CSPRNG, or it may be the case that the CSPRNG has, by some design decision, been given less priority

regarding performance. The performance penalty in these instances is greater for the Knuth-Yao convolution method, than it is for our sampler. This is by a factor of 16 for $256 < \sigma < 65536$ and a factor of 32 for $\sigma > 65536$. The results of Table 2 were performed with an Intel hardware-accelerated CSPRNG and, hence, such a penalty is minimised.

The work in this paper serves as a starting point for research into rounded, continuous Gaussian sampling for lattice-based signatures. Whether this be as a benchmark for different avenues of exploration, or as the basis on which to enhance the performance of the CORDIC implementation, the contributions in this paper outline a number of useful techniques and results for evaluating side-channel-secure rounded Gaussian samplers.

References

1. Kuznetsov, A., Svatovskij, I., Kiyan, N., Pushkar'ov, A.: Code-based-public-key cryptosystems for the post-quantum period. In: 2017 4th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), pp. 125–130. IEEE (2017)
2. Finiasz, M., Sendrier, N.: Security bounds for the design of code-based cryptosystems. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 88–105. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_6
3. Mozaffari-Kermani, M., Azarderakhsh, R.: Reliable hash trees for post-quantum stateless cryptographic hash-based signatures. In: 2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pp. 103–108. IEEE (2015)
4. Ding, J., Petzoldt, A.: Current state of multivariate cryptography. *IEEE Secur. Priv.* **15**(4), 28–36 (2017)
5. Azarderakhsh, R., et al.: Supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Standardization Project (2017)
6. Seo, H., Liu, Z., Longa, P., Hu, Z.: SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 1–20 (2018)
7. Bos, J., et al.: Frodo: take off the ring! practical, quantum-secure key exchange from LWE. *Cryptology ePrint Archive*, Report 2016/659 (2016). <https://eprint.iacr.org/2016/659>
8. Bos, J., et al.: CRYSTALS - Kyber: a CCA-secure module-lattice-based KEM. *Cryptology ePrint Archive*, Report 2017/634 (2017). <https://eprint.iacr.org/2017/634>
9. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS - Dilithium: digital signatures from module lattices. *IACR Cryptology ePrint Archive*, vol. 2017, p. 633 (2017)
10. Peikert, C.: A decade of lattice cryptography. *Found. Trends® Theor. Comput. Sci.* **10**(4), 283–424 (2016). <https://doi.org/10.1561/04000000074>
11. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, ser. STOC 2008, pp. 197–206. ACM, New York (2008). <https://doi.org/10.1145/1374376.1374407>

12. Chen, J., Lim, H.W., Ling, S., Wang, H., Nguyen, K.: Revocable identity-based encryption from lattices. In: Susilo, W., Mu, Y., Seberry, J. (eds.) ACISP 2012. LNCS, vol. 7372, pp. 390–403. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31448-3_29
13. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. In: 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 372–381 (October 2004)
14. Groot Bruinderink, L., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload – a cache attack on the BLISS lattice-based signature scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 323–345. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_16
15. Fouque, P.-A., et al.: Falcon: fast-Fourier lattice-based compact signatures over NTRU. Submission to the NIST’s Post-Quantum Cryptography Standardization Process (2018)
16. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. In: IACR Cryptology ePrint Archive (2015)
17. Zhao, R.K., Steinfeld, R., Sakzad, A.: FACCT: fast, compact, and constant-time discrete gaussian sampler over integers. *IEEE Trans. Comput.* **69**(1), 126–137 (2019)
18. Karmakar, A., Roy, S.S., Reparaz, O., Vercauteren, F., Verbauwhede, I.: Constant-time discrete gaussian sampling. *IEEE Trans. Comput.* **67**(11), 1561–1571 (2018)
19. Micciancio, D., Walter, M.: Gaussian sampling over the integers: efficient, generic, constant-time. Tech. Rep. 259 (2017). <https://eprint.iacr.org/2017/259>
20. Hülsing, A., Lange, T., Smeets, K.: Rounded Gaussians. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. LNCS, vol. 10770, pp. 728–757. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76581-5_25
21. Andryscio, M., Nötzli, A., Brown, F., Jhala, R., Stefan, D.: Towards verified, constant-time floating point operations. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS 2018, pp. 1369–1382. ACM, New York (2018). <https://doi.org/10.1145/3243734.3243766>
22. Box, G.E.P., Muller, M.E.: A note on the generation of random normal deviates. *Ann. Math. Stat.* **29**, 610–611 (1958)
23. Walther, J.S.: A unified algorithm for elementary functions. In: Proceedings of the May 18–20, 1971, Spring Joint Computer Conference, ser. AFIPS 1971 (Spring), pp. 379–385. ACM, New York (1971). <https://doi.org/10.1145/1478786.1478840>
24. Boppana, L., Dhar, A.: CORDIC architectures: a survey. *VLSI Des.* **2010**, 03 (2010)



Accelerating Lattice Based Proxy Re-encryption Schemes on GPUs

Gyana Sahu^(✉)  and Kurt Rohloff^(D) 

New Jersey Institute of Technology, Newark, NJ 07102, USA
{grs22,rohloff}@njit.edu

Abstract. Proxy Re-Encryption (PRE) is an indispensable tool in many public-key cryptographic schemes that enables users to delegate decryption rights to other users via a proxy. In this work, we present a high performance implementation of PRE schemes on NVIDIA GPUs. We target two lattice based PRE schemes, BV-PRE and Ring-GSW PRE defined over polynomial rings. We design a parallel Number Theoretic Transform (NTT) procedure capable of working on arbitrary precision moduli (in CRT form) and demonstrate several low level and GPU optimizations techniques to accelerate the PRE schemes.

For the same or higher security settings our results show 39x to 228x factors of improvement in performance with a peak throughput of 6.3 Mbps when compared to the CPU implementation of the BV-PRE scheme in the PALISADE lattice crypto software library. Similarly, for the Ring-GSW PRE scheme we achieve a peak throughput of 49 Mbps and up to 11x improvement in performance.

Keywords: Homomorphic encryption · Ring-LWE · GPU acceleration · CUDA

1 Introduction

First introduced in the work of Blaze, Bleumer and Strauss [3], Proxy Re-Encryption (PRE) is a powerful cryptographic primitive that allows a subscriber (Bob) to exchange and interpret encrypted data received from a publisher (Alice) without ever exchanging any secret key. To interpret the messages, Alice creates and gives to a Proxy (Polly) a re-encryption key which then allows Polly to transform messages encrypted with Alice's public key into an encrypted message that can be decrypted by Bob's secret key. Furthermore, semantic security of proxy re-encryption guarantees that the proxy Polly, does not learn anything about Alice's secret key or messages.

Proxy re-encryption can be a useful tool in brokering information exchanges in untrusted environments such as cloud computing platforms. Users can choose to store contents in encrypted form on the cloud and then register re-encryption keys of other users they want to interact with. Now, the cloud acting as a proxy can re-encrypt messages on the fly and deliver encrypted messages that can be

read by the desired users. Even if the cloud is corrupted by a malicious adversary and all the data stored on the cloud is compromised, the adversary cannot retrieve meaningful information out of it. Further, an adversary in possession of re-encryption keys cannot deduce the secret keys of either the producer or consumer. Additionally, if the PRE scheme in deployment is *key private* secure then the adversary cannot even trace the identities of Alice and Bob. It can be easily seen that a PRE scheme is not just restricted to cloud computing environments but the same idea can be extended to other similar privacy concerning applications. PRE schemes have been proposed for use in digital rights management (DRM) systems [24], secure file storage systems [2], email list services [17], and many other applications.

In the literature of cryptology many proxy re-encryption schemes have been proposed starting with the pioneering work of the BBS [3] PRE scheme. Other PRE schemes have been known to be constructed on bilinear pairings and the decisional Diffie-Hellman (DBDH) assumption. Many of these schemes are computationally intensive and inefficient to implement in real world applications. Another problem associated with some of these PRE schemes is their *bi-directional* nature which allows them to perform re-encryption from publisher encrypted data to consumer encryption and also in reverse using the same re-encryption key. This is considered to be an undesirable property because of unwarranted rights acquired by the proxy. A *uni-directional* PRE scheme is more practical as it possesses translation capability in one direction only. Another important property that defines the flexibility of a PRE scheme is the number of hops for which it can be re-encrypted. A *multi-hop* PRE scheme was first presented by Canneti and Hohenberger [7]. Among the many open problems cited in their work was the construction of a PRE scheme which is simultaneously *multi-hop* and *uni-directional*.

The emergence of lattice based cryptography has paved the road for many new constructions with some of them being quite efficient and secure against quantum computers. Among the most notable developments is the breakthrough work of Gentry [14] on fully homomorphic encryption. A subsequent line of work by Brakerski and Vaikuntanathan [4–6] presented FHE schemes based on standard lattice assumptions which can be reduced to worst case hardness of approximating lattice problems. These FHE schemes are endowed with much better noise controlling mechanisms which allow them to evaluate circuits of greater depth. However, to keep the noise growth to a minimum, multiplication is often performed using a binary multiplication tree. Gentry *et al.* [13] introduced a FHE scheme roughly similar to Regev's [23] encryption scheme which further improved upon this noise growth during circuit evaluation by restricting it to a quasi additive nature. Such asymmetric noise growth can be used to support sequential multiplication of ciphertexts.

Building a PRE primitive with these FHE schemes presents a simple yet powerful approach for extending them for information exchanges in an untrusted environment. Further, these PRE schemes mitigate some of the above mentioned problems associated with traditional PRE schemes as they are inherently

endowed with a *uni-directional* and *multi-hop* nature. A construction of a PRE scheme based on the BV [5] FHE scheme was presented in [22] where the authors exploited the key-switching procedure for achieving the re-encryption functionality. Along the same lines, we extended the PRE scheme to the Ring-GSW [16] FHE scheme. Evaluation of both BV-PRE and Ring-GSW-PRE schemes (with $r = 1$) with standard parameter set and security factor (100-bits) shows that the re-encryption procedure can be completed in the order of 10–100 ms.

Nowadays, owing to the ever increasing computational power and network efficiency most of the applications hosted on the cloud are expected to work in real time. An application involving a PRE primitive is no different. Being a low level primitive, the PRE scheme in deployment is expected to deliver the least possible latency. The above mentioned PRE schemes share the similarity of using the algebraic structure of ideal lattices as polynomial rings (RLWE). As a result, many of the computational bottlenecks in the implementation of these PRE schemes arise due to large ring dimensions, arbitrary precision multiplications, polynomial tensors, number theoretic transforms, matrix multiplication, etc. Over the years many of these problems have been remedied by switching over to better algorithms or optimizations. However, to provide additional speedups better hardware architectures have to be considered. In a heterogeneous computing model, these hardware accelerators acting as co-processors can be orchestrated by CPUs to achieve the desired levels of throughput. Among the common hardware accelerators such as FPGAs, ASICs and GPUs the most common and readily available solution is provided by GPUs. Modern GPUs consist of streaming multi-core processors which can be utilized to accelerate parts of computations that can be processed in parallel.

Lattice based cryptography and more specifically RLWE based FHE schemes being amenable to such parallelism have shown significant folds of performance improvement when implemented on GPUs. For example, in [8] the authors presented an implementation of the NTRU FHE scheme on GPUs and evaluated the AES and Prince block ciphers resulting in 2.5–7.6x factors of speedup over CPUs. Similarly, in [9] the authors present a homomorphic encryption accelerator library, cuHE targeting LTV [20], BGV [4] and DHS [11] FHE schemes. Speedups of 12–41 were reported for homomorphic sorting of ciphertexts. In another work, [16] leveraged the power of GPUs towards constructing a homomorphic Bayesian spam filter, secure multiple keyword search and evaluation of binary decision trees based on the Ring-GSW FHE scheme. For the same security settings [16] reported a factor of 10x improvement in performance when compared with the IBM HeLib [15] software library. Continuing in this line of work, we present an implementation of PRE schemes based on BV and Ring-GSW FHE schemes.

Our Contributions: We enumerate our main contributions and scope of the paper as follows.

- We present a GPU implementation of number theoretic transforms (NTT) based on finite field arithmetic where butterfly operations are performed in parallel. In order to extend the finite field over large numbers we keep integers

in CRT representation. Our NTT implementation targets both small ($n \leq 1024$) and large ($n > 1024$) polynomial dimensions.

- Armed with a parallel implementation of NTT operations, next we target the parallelization of bit decomposition procedure. Relinearization along with bit/digit decomposition is considered to be the most critical procedure in many homomorphic encryption schemes and accelerating this operation imparts overall efficiency to the FHE scheme.
- Finally, utilizing the above implementations we demonstrate the acceleration of BV-PRE and Ring-GSW PRE schemes. In our implementation, we have reduced the number of memory transfers between host and device to a minimum by storing most of the dynamic elements on GPU memory. Another key feature of our implementation is the use of CUDA streams which allow concurrent execution of kernels thereby minimizing latency.

Paper Organization: In Sect. 2, we first provide the basic syntax of a public-key encryption scheme augmented with proxy re-encryption procedures. Section 3 introduces the mathematical notations and preliminaries we have used throughout the paper. Section 4 discusses the implementation of the underlying arithmetic layer, number theoretic transforms and bit decomposition procedures on the GPU along with other optimizations. The next two sections, Sects. 5 and 6 describe the lattice-based PRE schemes. In Sect. 7, we provide the parameters selected for implementation. Section 8 discusses the evaluation of PRE-schemes and their overall speedups.

2 Design

2.1 Syntax of Unidirectional PRE Scheme

We recall that a non-interactive PRE scheme is an ensemble of PPT algorithms $\Pi = (\text{ParamsGen}, \text{KeyGen}, \text{ReKeyGen}, \text{Encrypt}, \text{ReEncrypt}, \text{Decrypt})$, which can be defined as per the following syntax:

- **ParamsGen**(1^λ): It takes the security parameter λ and returns the corresponding public parameters pp .
- **KeyGen**($pp, 1^\lambda$): It takes the public parameters pp and returns the key pair (pk, sk) .
- **ReKeyGen**(pp, sk_i, pk_j): It takes the public parameters, secret key of publisher i , public key of subscriber j and returns the re-encryption key $rk_{i \rightarrow j}$.
- **Encrypt**(pp, pk, m): Given public key and public parameters, it encrypts the message m and returns a ciphertext c .
- **ReEncrypt**($pp, rk_{i \rightarrow j}, c_i$): It transforms a ciphertext c_i of the party i into a ciphertext c_j that can be decrypted by party j .
- **Decrypt**(pp, sk, c): It recovers the message m from ciphertext c .

3 Preliminaries and Mathematical Notations

In this work, we represent scalars in plain, vectors by lower-case bold letters (e.g., \mathbf{a}) and matrices by upper-case bold letters (e.g., \mathbf{A}). Ring elements, for example $x \in R_q$, are represented by lower case letters in plain. We define $[k] = \{1, \dots, k\}$ for any non-negative integer k . The i -th norm of a vector \mathbf{v} is denoted by $\|\mathbf{v}\|_i$. We denote the tensor (Kronecker) product of two matrices \mathbf{A} and \mathbf{B} as $\mathbf{A} \otimes \mathbf{B}$. We extend this notation to a vector \mathbf{v} where the Kronecker product is represented as $\mathbf{v} \otimes \mathbf{A}$. Unless explicitly mentioned logarithms are to be understood with base 2. We denote the horizontal and vertical concatenation of matrices by operators $(\cdot ||)$ and $(\cdot ||^\top)$ respectively. For two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}^{n \times n}$ $[\mathbf{A} || \mathbf{B}]$ produces a matrix $\mathbf{C} \in \mathbb{Z}^{n \times 2n}$. Similarly, $[\mathbf{A} ||^\top \mathbf{B}]$ produces a matrix $\mathbf{C} \in \mathbb{Z}^{2n \times n}$.

3.1 Gadget Matrix and Relinearization Functions:

For LWE dimension n and modulus q , we use the following “gadget” vector [21]:

$$\mathbf{g} = (1, 2, 4, \dots, 2^{\ell-1}) \in \mathbb{Z}_q^\ell, \text{ where } \ell = \lceil \log q \rceil.$$

The *gadget matrix* \mathbf{G} is then defined as the diagonal concatenation of the \mathbf{g} vector n times. Formally, it is written as $\mathbf{G} = \mathbf{g} \otimes I_n \in \mathbb{Z}_q^{\ell n \times n}$. To perform relinearization we define the following operations [5, 13] with respect to an element a which can be either a vector or matrix or polynomial ring.

- **BitDecomp** (a): For a vector $\mathbf{a} \in \mathbb{Z}_q^n$ this operation produces a bit decomposed and expanded vector $\mathbf{a}' \in \mathbb{Z}_q^{n\ell}$ where $a_i = \sum_{j=0}^{\ell-1} 2^j a'_{i\ell+j}$. In case of a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ this operation produces a matrix that is expanded along the column resulting in $\mathbf{A}' \in \mathbb{Z}_q^{m \times n\ell}$. Finally, in case of a polynomial ring $a \in R_q$ this operation produces $\mathbf{a}' \in R_q^\ell$.
- **PowerOf2** (a): Given a vector $\mathbf{a} \in \mathbb{Z}_q^n$ this operation produces an expanded vector $\mathbf{a}' \in \mathbb{Z}_q^{n\ell}$ where $\mathbf{a}' = (a_0, 2a_0, \dots, 2^{\ell-1}a_0, \dots, 2^{\ell-1}a_{n-1})$. Similarly, for a polynomial ring $a \in R_q$ we get $\mathbf{a}' \in R_q^\ell$ where $a'_i = 2^i a$.

Using these operations, we can produce a product of \mathbf{a} and \mathbf{b} as follows:

$$\langle \mathbf{BitDecomp}(\mathbf{a}), \mathbf{PowersOf2}(\mathbf{b}) \rangle = \langle \mathbf{a}, \mathbf{b} \rangle$$

4 Number Theoretic Transform and Bit-Decomposition

4.1 Number Theoretic Transform

Cryptosystems based on the RLWE security assumption are defined over a polynomial ring $R = \mathbb{Z}[X]/\Phi_m(X)$ where $\Phi_m(X)$ is an irreducible monic cyclotomic polynomial of order m . This notation is extended to a polynomial R_q modulo an integer q where the coefficients of the polynomial are in the interval $(-q/2, q/2]$. Alternatively, an element $a \in R_q$ is simply considered to be a coefficient vector

$\mathbf{a} \in \mathbb{Z}_q^{\varphi(m)}$. While addition of these polynomials is quite efficient, multiplication leads to quadratic time complexity. To circumvent this inefficiency, we represent polynomial rings in the so called “Evaluation” representation. For a polynomial $a \in R_q$ the coefficients can be converted to the evaluation domain \bar{a} by evaluating $a(X)$ at each of the m -th primitive roots of unity modulo q . The coefficients of \bar{a} are related to polynomial a through the relation $\bar{a}_i = a(\omega^i) \pmod q$ where $(i, m) = 1$ and ω is a m -th primitive root of unity modulo q .

This back and forth conversion of a polynomial can be achieved efficiently by using number theoretic transforms (NTT) which is roughly similar to the classical n -dimensional fast Fourier transform where a finite field is used instead of complex numbers. Concretely, in our implementation, we use the power of two cyclotomics ($m = 2^k$) where $\Phi_m(X)$ is maximally sparse and the ring dimension $n = \varphi(m) = m/2$ is also a power of two. The power of two cyclotomics along with NTT has become so pervasive in lattice based cryptography that the overall efficiency of the cryptosystem depends upon the latency of NTT procedure. For this reason, we chose to implement NTT as an iterative Cooley-Tukey algorithm. More specifically, we implemented the NTT routine with Fermat theoretic transform (FTT) optimization which eliminates interleaved zero paddings when using the conventional NTT procedure.

4.2 Parallel NTT

Exploiting the NVIDIA GPU architecture, we reduce the NTT latency further by mapping the butterfly computations of each of the $\log n$ stages to an independently processing thread of a thread block. In the NVIDIA CUDA architecture, each kernel or device function can be potentially divided into a 3-dimensional array of blocks where a block further consists of several threads. Because of hardware restrictions, a maximum of 1024 threads can be assigned to a block. Further, the threads within a block have the capability to share data and more importantly synchronize with each other. In our implementation, for small polynomials ($n \leq 1024$) we map the coefficients entirely to a thread block and synchronize the thread block after completion of a stage as shown in Fig. 1. We use shared memory for storing the intermittent results as latency associated with global memory is higher than that of shared memory which resides on the chip. For larger polynomial rings ($n > 1024$), we use a combination of block level synchronization and stream level synchronization to avoid data race conditions. As stream level synchronization avoids data race conditions via global memory synchronization we pay the penalty of using slower memory but only for a fraction of the NTT procedure calls.

The evaluation of the proposed NTT procedure on GPU platforms and CPU platforms is shown in Fig. 2. From the figure, we can see that the CPU platform running on a single thread achieves slightly better performance for smaller ring dimensions. As the ring dimensions grow higher we can see that the GPU platform starts showing significant improvement in performance.

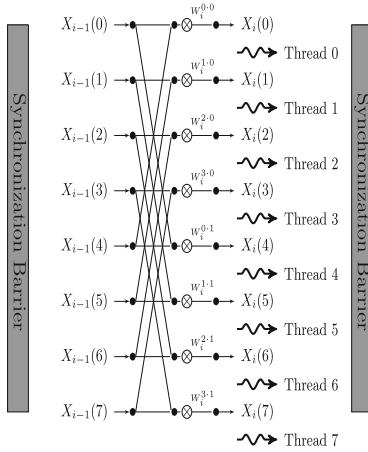


Fig. 1. Parallel implementation of the i -th stage of NTT on a GPU, $N = 8$.

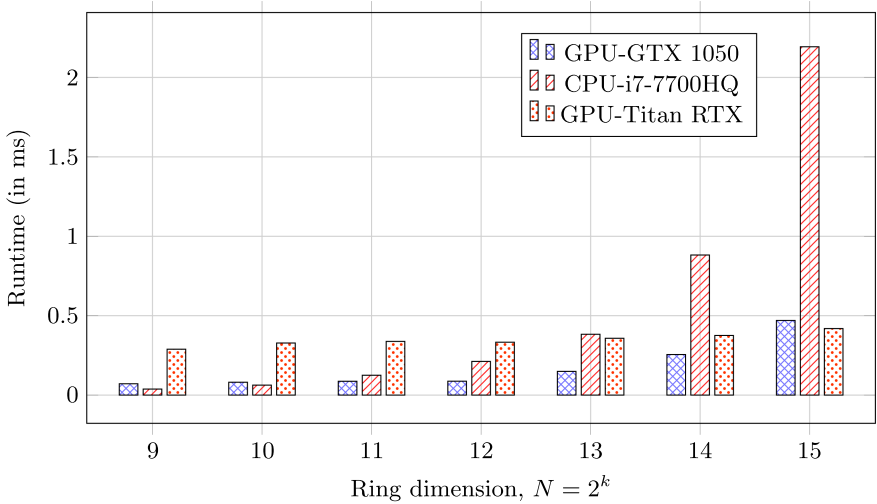


Fig. 2. Comparison of CPU and GPU runtimes of the NTT algorithm.

4.3 Barrett Modulo Reduction and Arbitrary Precision Support

For modulo reduction, we used a variation of the generalized Barrett modulo reduction [10] as outlined in Algorithm 1. Barrett modulo reduction requires a pre-computation term $\mu = \lfloor 2^{2b}/q \rfloor$ for a particular modulus q and its bit width, $b = \lceil \log_2 q \rceil$. We pre-compute these terms and transfer them to GPU global memory for read only access to any kernel. NVIDIA GPUs are restricted to a 32-bit architecture and 64-bit arithmetic are supported only through assembly code emulation. For this reason, we prefer moduli with bit width closer to 32-bits

so that the precomputation term μ fits into the word size. Concretely, we allow up to 29–30 bit width moduli in our implementation.

Lattice based cryptosystems employ the addition of low norm noise terms to base their security on Ring-LWE and LWE assumptions. For preserving the correctness constraints so that ciphertexts are decrypted correctly, the modulus q should be chosen large enough such that the final accumulated error terms do not “wrap around” modulo q . To extend support for larger moduli we store a set of increasing prime moduli q_i by an application of the Chinese Remainder Theorem (CRT). We only reconstruct the coefficients into larger terms for the purpose of decryption or bit-decomposition where the polynomial needs to be represented in terms of the larger modulus, $q = \prod_{i=0}^{t-1} q_i$.

Evaluation of the NTT procedure on a GPU with varying number of moduli, t and ring dimension N can be seen in Fig. 3. For most of the ring dimensions, the runtimes vary a little. This is due to the fact that GPUs have the capability to improve throughput by hiding latency with the concurrent execution of the NTT procedure on different polynomials. On a CPU platform the runtime is estimated to scale linearly with the number of moduli assuming a single thread execution environment.

Algorithm 1: Mod Barrett Reduction

Input : $x \in [0, (q - 1)^2]$, modulus q , bit-width $b = \lceil \log q \rceil$, $\bar{q} = 2 \cdot q$ and $\mu = \lfloor 2^{2b}/q \rfloor$.

Output: $z \leftarrow x \% q$

```

1  $z \leftarrow x \gg b$  ;
2  $z \leftarrow z \cdot \mu$  ;
3  $z \leftarrow z \gg b$  ;
4  $z \leftarrow z \cdot q$  ;
5  $x \leftarrow x - z$  ;
6 if  $x \geq \bar{q}$  then
7   |  $z \leftarrow x - \bar{q}$ ;
8 end if
9 if  $z \geq q$  then
10  |  $z \leftarrow z - q$ ;
11 end if
12 return  $z$ 
```

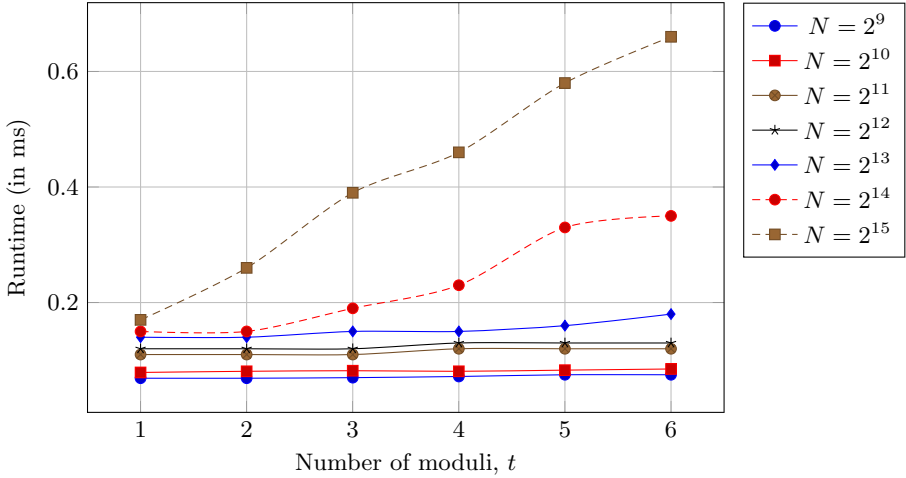


Fig. 3. GPU runtimes of the NTT algorithm with varying dimension N and moduli t .

4.4 Bit Decomposition

Bit decomposition along with the relinearization procedure forms the backbone of lattice based cryptography. While bit decomposing integers is simple in finite field arithmetic, it is accompanied with additional overheads in Ring-LWE based cryptosystems. In Ring-LWE cryptosystems, ciphertexts and other key elements are mostly present in evaluation representation. To bit decompose, polynomial rings need to be switched back to coefficient representation. At this stage, bit decomposition of the polynomial results in a vector of b polynomials, where b is the bit length of the modulus q . To perform further computations, these polynomials need to be converted back to evaluation representation by a series of NTT calls. Since the bit decomposed polynomials are independent of each other, we apply NTT procedures on them in parallel using a three dimensional CUDA grid mapping. To avoid race conditions in the NTT procedure, we provide the kernel with appropriate synchronization. From Fig. 4, we can observe that our GPU implementation of bit decomposition outperforms runtimes of the CPU platform for all ring dimensions and further the speedups are more pronounced in case of higher ring dimensions.

5 PRE Cryptosystem with BV FHE Scheme

The BV-PRE [22] scheme described here is based on the BV FHE [5] scheme introduced by Brakerski and Vaikuntanathan. The message space for the scheme is restricted to $\mathcal{M} \in R_p$ where p is the plaintext modulus.

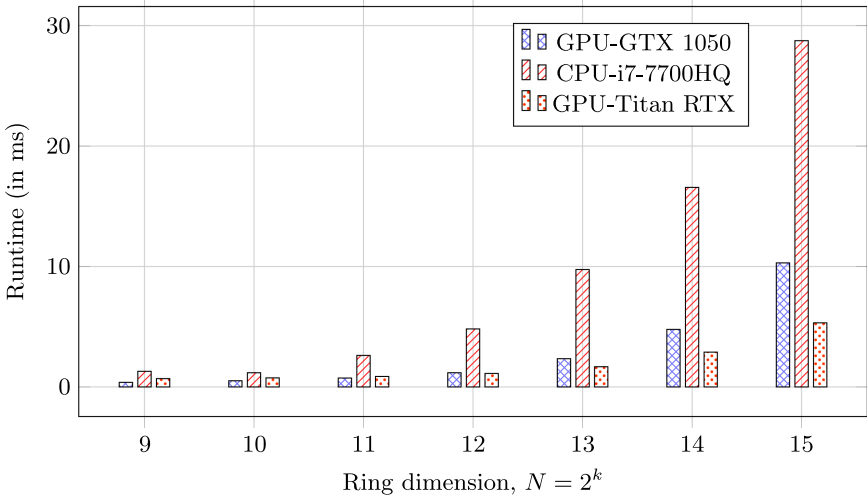


Fig. 4. GPU vs CPU runtimes of bit decomposing a polynomial ring with varying ring dimension N .

5.1 BV Encryption Scheme

The scheme is parameterized using the following quantities:

- Security parameter (λ) ,
- Ciphertext modulus q and plaintext modulus $2 \leq p \ll q$,
- Ring dimension n ,
- D -bounded discrete Gaussian error distribution χ_e with distribution parameter σ_e ,
- Ternary uniform distribution \mathcal{T} which samples from $\{-1, 0, 1\}$,
- Discrete uniform distribution \mathcal{U}_q ,

The scheme consists of the following operations:

- **ParamsGen** (1^λ) : Choose positive integers $q = q(\lambda)$ and $n = n(\lambda)$. Return $pp = (p, q, n)$.
- **KeyGen** (pp, λ) : Sample polynomials $a \leftarrow_s \mathcal{U}_q$, $s \leftarrow_s \mathcal{T}$ and $e \leftarrow_s \chi_e$. Compute $b := a \cdot s + pe \in R_q$. Set the public key pk and private key sk as follows:

$$sk := (1, s) \in R_q^2, pk := (a, b) \in R_q^2$$

- **Encrypt** $(pp, pk = (a, b), m \in \mathcal{M})$: Sample polynomials $v \leftarrow_s \mathcal{T}$, $e_0, e_1 \leftarrow_s \chi_e$. Compute the ciphertext $c = (c_0, c_1) \in R_q^2$ as follows:

$$c_0 = b \cdot v + pe_0 + m \in R_q, c_1 = a \cdot v + pe_1 \in R_q$$

- **Decrypt** $(pp, sk = s, c = (c_0, c_1))$: Compute the ciphertext error $t = c_0 - s \cdot c_1 \in R_q$. Output $m' = t \bmod p$. For correct decryption, the coefficients of the noise polynomial, t should not wrap around modulo q .

5.2 Proxy Re-encryption Scheme

We refer to the publisher of information as party A and the subscriber of information via the proxy as party B. Additional operations pertaining to PRE computation are as follows:

- **Preprocess**($pp, \lambda, sk_B = (1, s_B)$) Sample uniformly distributed random polynomials $\alpha_i \leftarrow_s \mathcal{U}_q$ and error polynomials $e_i \leftarrow_s \chi_e$ for $i \in [0, \lceil \log_2(q)/r \rceil]$. Here r is the relinearization window. Compute the following elements:

$$\gamma_i = \alpha_i \cdot s_B + pe_i \in R_q; pk_B = (\alpha_i, \gamma_i)_{i \in \{0, 1, \dots, \lceil \log_2(q)/r \rceil\}}$$

- **ReKeyGen**($pp, sk_A = (1, s_A), pk_B$): Compute β_i and set the re-encryption key as follows:

$$\beta_i = \gamma_i - s_A \cdot (2^r)^i \in R_q, rk_{A \rightarrow B} = (\alpha_i, \beta_i)_{i \in \{0, 1, \dots, \lceil \log_2(q)/r \rceil\}}$$

- **ReEncrypt**($pp, rk_{A \rightarrow B} = (\alpha_i, \beta_i), c = (c_0, c_1)$): To get the re-encrypted ciphertext $c' = (c'_0, c'_1)$ first apply 2^r base decompositions and proceed as follows:

$$c'_0 = c_0 + \sum_{i=0}^{\lceil \log_2(q)/r \rceil} (c_1^{(i)} \cdot \beta_i), c'_1 = \sum_{i=0}^{\lceil \log_2(q)/r \rceil} (c_1^{(i)} \cdot \alpha_i)$$

where $c_1^{(i)}$ represents the i -th digit of the base- 2^r decomposition of c_1 .

Under the new secret key $sk = (1, s_B)$ decryption of the ciphertext $c' = (c'_0, c'_1)$ can be shown as

$$\begin{aligned} c'_0 - s_B \cdot c'_1 &= c_0 + \sum_{i=0}^{\lceil \log_2(q)/r \rceil} (c_1^{(i)} \cdot \beta_i) - s_B \cdot \sum_{i=0}^{\lceil \log_2(q)/r \rceil} (c_1^{(i)} \cdot \alpha_i) \\ &= c_0 + \sum_{i=0}^{\lceil \log_2(q)/r \rceil} (c_1^{(i)} \cdot \{ \alpha_i \cdot s_B + pe_i - s_A \cdot (2^r)^i \}) \\ &\quad - s_B \cdot \sum_{i=0}^{\lceil \log_2(q)/r \rceil} (c_1^{(i)} \cdot \alpha_i) \\ &= c_0 - s_A \cdot c_1 + pE_i; E_i = \sum_{i=0}^{\lceil \log_2(q)/r \rceil} (c_1^{(i)} \cdot e_i) \end{aligned}$$

To preserve the correctness of the re-encrypted ciphertext $c' = (c'_0, c'_1)$, we derive the following constraint:

$$\begin{aligned} \|t\|_\infty &\leq 3\sqrt{n}pD + \sqrt{n}pD \cdot D_r \lceil \log_2(q)/r \rceil \\ &\approx \sqrt{n}pDD_r (\lceil \log_2(q)/r \rceil + 1) \ni r > 1 \\ &\Rightarrow q \geq 2\sqrt{n}pDD_r (\lceil \log_2(q)/r \rceil + 1) \end{aligned}$$

where, D_r is the bound of the bit decomposed polynomial in base 2^r representation.

5.3 Security

The security of PRE schemes is generally covered under indistinguishability against chosen-plaintext attacks (IND-CPA). For brevity we omit the formal definition of the IND-CPA security notion and capture the security with the following theorem.

Theorem 1 [22]. *Under the $\mathbf{RLWE}_{\Phi, q, \chi_e}$ assumption, the BV-PRE scheme is IND-CPA secure. Specifically, for a poly-time adversary \mathcal{A} , there exists a poly-time distinguisher \mathcal{D} such that*

$$Adv_{\mathcal{A}}^{cpa}(\lambda) \leq (\rho \cdot (Q_{rk} + Q_{re}) + N + 1) \cdot Adv_{\mathcal{D}}^{\mathbf{RLWE}_{\Phi, q, \chi_e}}$$

where Q_{rk} and Q_{re} are the numbers of re-encryption key queries and re-encryption queries, respectively; N is the number of honest entities; λ is the security parameter; Φ is the cyclotomic polynomial defining the ring $R_q = \mathbb{Z}_q[x]/\langle \Phi \rangle$ and $\rho = \lceil \log_2(q)/r \rceil$.

6 PRE Cryptosystem with Ring-GSW FHE Scheme

The Ring-GSW PRE scheme described here is based on the Ring-GSW FHE scheme, a RLWE variant of the GSW [13] FHE scheme. The message space for the scheme is restricted to $\mathcal{M} \in R_p$ similar to the BV-FHE scheme.

6.1 Ring-GSW Encryption Scheme

The scheme is parameterized similar to the BV-FHE scheme. It consists of the following operations:

- **ParamsGen**(1^λ): Choose positive integers $q = q(\lambda)$ and $n = n(\lambda)$. Return $pp = (\ell, N, p, q, h, n)$ where $\ell = \lceil \log_2(q)/h \rceil$ and $N = 2\ell$. Here h is the relinearization factor.
- **KeyGen**(pp, λ): Sample polynomials $a \leftarrow_s \mathcal{U}_q$, $s \leftarrow_s \mathcal{T}$ and $e \leftarrow_s \chi_e$. Compute $b := a \cdot s + pe \in R_q$. Set the public key pk and private key sk as follows:

$$sk := (1; -s) \in R_q^{2 \times 1}, \quad pk := \mathbf{A} = [a \ b] \in R_q^{1 \times 2}$$

- **Encrypt**($pp, pk = \mathbf{A}, m$): Sample random matrix \mathbf{R} from ternary distribution and an error matrix $\mathbf{E} \in R_q^{N \times 2}$ from discrete Gaussian distribution. Compute the ciphertext $\mathbf{C} \in R_q^{N \times 2}$ as follows:

$$\mathbf{R} = \{r_0, \dots, r_{N-1}\} \leftarrow_s \mathcal{T}_{R_q}^N, \quad \mathbf{E} \leftarrow_s \chi_{e, R_q}^{N \times 2}$$

$$\mathbf{C} = m \cdot \mathbf{G} + \mathbf{R} \otimes \mathbf{A} + p\mathbf{E}$$

- **Decrypt**($pp, sk = (1; -s), \mathbf{C}$): Message m' is recovered by multiplying the first row of the ciphertext \mathbf{C} with the secret key sk . This is shown as:

$$m' = (\mathbf{C}_0 \times sk \text{ mod } q) \text{ mod } p$$

Decryption works correctly as the first row of the ciphertext is in the form of a BV scheme ciphertext.

6.2 Proxy Re-encryption Scheme

We describe the operations pertaining to PRE computation for a publisher A and a subscriber B as follows:

- **ReKeyGen**(pp, sk_A, pk_B): The re-encryption key consists of two polynomial matrices. To generate the matrices, we first sample two ternary uniform matrices $\mathbf{R}_0, \mathbf{R}_1 \in R_q^N$. Next, we sample two error matrices from χ_{e, R_q} and set the evaluation matrices $\mathbf{EK}(i)$ as follows:

$$\begin{aligned} \mathbf{R}_i &\leftarrow_s \mathcal{T}_{R_q}^N, \quad \mathbf{E}_i \leftarrow_s \chi_{R_q, B}^{N \times 2}, \quad i \in \{0, 1\} \\ \mathbf{EK}[i] &= \mathbf{R}_i \otimes \mathbf{A}_B + p\mathbf{E}_i + (\text{PowerOf2}(sk_A) \ggg i) \\ rk_{A \rightarrow B} &= \{ \mathbf{EK}[0], \mathbf{EK}[1] \} \end{aligned}$$

- **ReEncrypt**($pp, \mathbf{C}_A, rk_{A \rightarrow B}$): We use the top ℓ rows of the ciphertext \mathbf{C}_A to perform re-encryption and denote this as $\mathbf{C}_A^{\text{top}}$. Next, we multiply each of the matrices $\mathbf{EK}[i]$ with $\mathbf{C}_A^{\text{top}}$ and reassemble the results into a Ring-GSW ciphertext $\mathbf{C}_{A \rightarrow B}$. This is shown as follows:

$$\begin{aligned} \mathbf{C}_{A \rightarrow B}^i &= \text{BitDecomp}(\mathbf{C}_A^{\text{top}}) \cdot \mathbf{EK}[i] \in R_q^{\ell \times 2} \\ \mathbf{C}_{A \rightarrow B} &= [\mathbf{C}_{A \rightarrow B}^0 \parallel^T \mathbf{C}_{A \rightarrow B}^1] \end{aligned}$$

To formulate the correctness constraint we have to ensure that there is no wrap around mod- q in the noise term t while decrypting the re-encrypted ciphertext. The noise term t is given by:

$$t = \mathbf{C}_{A \rightarrow B, 0} \times sk_B = \mathbf{C}_{A \rightarrow B, 0, 0} - s_B \mathbf{C}_{A \rightarrow B, 0, 1}$$

Let $\alpha_i = \text{BitDecomp}(\mathbf{C}_{A, j, 0}^{\text{top}})$ and $\beta_i = \text{BitDecomp}(\mathbf{C}_{A, j, 1}^{\text{top}})$ for $(i, j) \in [0, \ell)$. Then, $\mathbf{C}_{A \rightarrow B, 0}$ can be shown as:

$$\begin{aligned} \mathbf{C}_{A \rightarrow B, 0} &= [\alpha_i \ \beta_i]_{i=0}^{\ell-1} \cdot [r_j \mathbf{A}_B + p\mathbf{E}_j + \text{PowerOf2}(sk_A)] \\ &\text{where } i \in [0, \ell) \text{ and } j \in [0, 2\ell). \end{aligned}$$

$$\begin{aligned} \mathbf{C}_{A \rightarrow B, 0, 0} &= b_B \sum_{i=0}^{\ell-1} (\alpha_i r_i + \beta_i r_{\ell+i}) + p \sum_{i=0}^{\ell-1} (\alpha_i \mathbf{E}_{i, 0} + \beta_i \mathbf{E}_{\ell+i, 0}) \\ &+ \sum_{i=0}^{\ell-1} \alpha_i \text{PowerOf2}(1) + \sum_{i=0}^{\ell-1} \beta_i \text{PowerOf2}(-s_A) \\ &= b_B r'_0 + p\mathbf{E}'_{0, 0} + \alpha - s_A \beta \end{aligned}$$

Similarly,

$$\begin{aligned} \mathbf{C}_{A \rightarrow B, 0, 1} &= a_B \sum_{i=0}^{\ell-1} (\alpha_i r_i + \beta_i r_{\ell+i}) + p \sum_{i=0}^{\ell-1} (\alpha_i \mathbf{E}_{i, 1} + \beta_i \mathbf{E}_{\ell+i, 1}) \\ &= a_B r'_0 + p\mathbf{E}'_{0, 1} \end{aligned}$$

Therefore,

$$t = r'_0 (b_B - a_B s_B) + p (\mathbf{E}'_{0,0} - s_B \mathbf{E}'_{0,1}) + \alpha - s_A \beta \cong m \pmod p$$

For correct decryption $\|t\|_\infty \leq q/2$. By using the central limit theorem we arrive at the final correctness constraint:

$$\begin{aligned} \|t\|_\infty &\leq 2p\sqrt{n}D_h D \lceil \log_2(q)/h \rceil \cdot (2\sqrt{n} + 1) + 3p\sqrt{n}D \approx 6pnD_h D \lceil \log_2(q)/h \rceil \\ &\Rightarrow q \geq 12pnD_h D \lceil \log_2(q)/h \rceil \end{aligned}$$

6.3 Security

IND-CPA security of the Ring-GSW-PRE scheme is defined in a similar manner as in the BV-PRE scheme and only differs in the parameter of ρ which describes the number of RLWE samples in the re-encryption key.

Theorem 2. *Under the $\mathbf{RLWE}_{\Phi, q, \chi_e}$ assumption, the Ring-GSW PRE scheme is IND-CPA secure. Specifically, for a poly-time adversary \mathcal{A} , there exists a poly-time distinguisher \mathcal{D} such that*

$$Adv_{\mathcal{A}}^{cpa}(\lambda) \leq (\rho \cdot (Q_{rk} + Q_{re}) + N + 1) \cdot Adv_{\mathcal{D}}^{\mathbf{RLWE}_{\Phi, q, \chi_e}}$$

where Q_{rk} and Q_{re} are the numbers of re-encryption key queries and re-encryption queries, respectively; N is the number of honest entities; λ is the security parameter; Φ is the cyclotomic polynomial defining the ring $R_q = \mathbb{Z}_q[x]/\langle \Phi \rangle$ and $\rho := 4 \lceil \log_2(q) \rceil$.

7 Parameter Selection

Estimating parameters for LWE or RLWE based encryption schemes is a significantly challenging task. On one hand we have to ensure that the chosen parameters generate an underlying RLWE instance that is hard to solve as per known attacks while on the other hand we have to also meet the correctness constraint. The correctness constraint can be trivially achieved by choosing an arbitrarily large modulus q . However, such a strategy is not suitable for efficient implementation and further leads to insecure LWE instances. It was shown in [18] when the modulus is exponential in the LWE dimension, $q \geq 2^{\mathcal{O}(n)}$ and error distribution is narrow enough, the secret key can be recovered in polynomial time using standard lattice basis reduction algorithms. In order to layout concrete parameters, Lindner and Peikert [19] gave a heuristic relation that computes the runtime of BKZ lattice reduction algorithm for a particular root Hermite factor, δ . This is shown as:

$$\log_2(t_{BKZ}) \geq \frac{1.8}{\log_2(\delta)} - 110$$

Further, Gentry *et al.* [12] gave a relation which computes minimum LWE dimension secure for a particular modulus q , standard deviation of error distribution σ_e and root Hermite factor, δ . Combining the two we can express the minimum LWE dimension to support κ -bits of security as follows:

$$n \geq \frac{\log_2(q/\sigma_e)(\kappa + 110)}{7.2}$$

In particular, we used the runtime of BKZ2.0 [1] (considered to be a improved version of BKZ algorithm) given by the relation

$$\log_2(t_{\text{BKZ2.0}}) \geq \frac{0.009}{\log_2^2 \delta} - 27$$

Again, combining this with the minimum LWE dimension relation from [12] we arrive at our final security constraint as follows:

$$n \geq \frac{\log_2(q/\sigma_e)\sqrt{\kappa + 27}}{0.379}$$

In our implementation we targeted for $\kappa = 128$ -bits of security for both BV-PRE and Ring-GSW-PRE scheme. Using the correctness constraint from the respective schemes we generated the working modulus for various ring dimensions as shown in Table 1.

Table 1. Minimum modulus bits, b that satisfies 128-bits of security for the BV-PRE and Ring-GSW PRE scheme according to the respective correctness constraints and varying dimension, n . $D = 15$ and $\sigma = 4$.

n	b	
	BV-PRE scheme	Ring-GSW-PRE scheme
512	18	-
1024	18	25
2048	19	26
4096	19	27
8192	20	28
16384	20	29

8 GPU Implementation and Results

8.1 Software Implementation

We evaluated both the BV-PRE and the Ring-GSW-PRE scheme on two NVIDIA GPU devices, namely, GeForce GTX-1050 and Titan-RTX as shown

in Table 2. While GPU1 is a commodity grade notebook GPU, GPU2 is a much more powerful GPU targeted towards compute intensive applications. Our software implementation follows the modular structure of the PALISADE homomorphic encryption library separating crypto implementations from lower level math layers. Our implementation is compiled with the CUDA 10.0 NVCC compiler along with C++14 support. Our single threaded execution environment consists of 64-bit x86 architecture with operating system for GPU1 and GPU2 as Ubuntu 18.04 and Scientific Linux 6.10 (available on university HPC) respectively.

Table 2. Configuration and features of GPU with their corresponding CPU used for the evaluation of the PRE schemes.

Feature	CPU1	CPU2	GPU1	GPU2
Model	Intel i7-7700HQ	Intel Silver 4114	NVIDIA GeForce GTX 1050	NVIDIA Titan RTX
Cores	4	10	640	4608
Clock rate	2.8 GHz	2.2 GHz	1.49 GHz	1.77 GHz
Multiprocessors	-	-	5	72
RAM memory	16 GBytes	181 GBytes	4042 MBytes	24190 MBytes
CUDA capability	-	-	6.1	7.5

In our implementation we primarily aimed to improve the runtimes of operations pertaining to the encryption scheme and the PRE scheme by switching the CPU routines with CUDA kernel calls. We outline the main aspects of our GPU optimizations as follows:

- **Optimized pre-computation phase:** Our implementation consists of a pre-computation phase wherein we compute most of the cryptosystem parameters and NTT related parameters on the CPU and transfer them to GPU global memory. For faster memory transfers we use coalesced memory spaces which require fewer memory transfer calls. On the CPU side we mostly use memory allocation using pinned host memory. Memory transfer using pinned memory is faster than pageable host memory because the GPU can directly access such memory spaces. Lastly, we remark that we do not make use of constant or texture memory as it reported little or no improvement in performance. Moreover, such memory spaces are available in very limited number and not scalable for higher ring dimensions.
- **Memory related optimizations:** Being a heterogeneous platform, one should expect a significant amount of data transfers between CPU and GPU memory. However, it is known that such data transfers are generally slower (because of lower bandwidth PCIe) and can sometimes degrade the overall performance of an application. To get a more realistic performance estimate of the various operations, we eliminated most of the data transfers by allocating memory for most of the cryptosystem elements on GPU memory directly.
- **Fast Random Generators:** Our implementation relies on the CUDA random number generation library cuRAND for generating random polynomials. The distributions targeted in our application are uniform random distribution U_q , discrete Gaussian distribution χ_e with standard deviation σ_e , binary

and ternary uniform distributions. Except for uniform distribution, all other distributions were generated using continuous Gaussian distribution on pre-allocated memory and then launching appropriate kernels to reduce them in a particular range. More specifically, we used the CURAND RNG PSEUDO MTGP32 set of generators which is 5x faster than other random number generators of the same family and atleast 10x faster than CPU random number generators. We remark that we have not evaluated the cryptographic security of this random number generator and assume that it generates long enough internal states with desirable statistical properties which prevents any adversarial attack.

- **Relinearization:** As described in Sect. 4.4 we benefit significantly from mapping the bit-decomposed polynomials to a three dimensional CUDA kernel wherein the NTT procedure can be invoked in parallel. Once the polynomials are evaluated they need to be reduced back into a single polynomial. To do this efficiently, we launch a parallel reduction kernel with grid dimension roughly equal to the number of bit-decomposed polynomials. After each kernel call, we obtain the partial results in half the polynomials we start with. Repeating this process until we reach a single reduced polynomial makes the relinearization process very efficient with only $\mathcal{O}(\log(\log(q)))$ reduction kernel calls.
- **Streams:** On GPUs we can increase the throughput of kernels by launching them simultaneously on independent streams. For example, NVIDIA K20 has the ability to support upto 32 concurrent kernels launched on a separate stream. In our implementation, we use streams mainly for the parallel generation of noise, synchronizing NTTs, asynchronous memory transfers and kernel parallelization. For taking advantage of stream APIs, we always keep the ciphertext components independent; for example in the Ring-GSW PRE scheme we keep the ciphertext as a vector of column polynomials in row major order.

8.2 Experimental Results

For experimental analysis of our PRE schemes, we use latency and throughput as primary yardsticks. A PRE scheme can be divided into a static and dynamic phase. The static phase takes into account all sorts of key generation, pre-computations and parameter setup. The performance of any real time system that uses a PRE scheme is largely determined by the dynamic phase where computations are performed on the fly. Therefore, we only report the runtimes and throughput for dynamic phase operations by varying the ring dimension, n and modulus bit length, b determined as per 128-bit security setting. For recording throughput, we consider messages as binary string with length equal to the ring dimension.

From Tables 3 and 4 we can observe that encryption and decryption runtimes for the BV-PRE scheme vary in a very small amount with increasing ring dimensions. We remark that encryption runtimes for smaller ring dimensions are still

slower with break-even occurring for $n = 2048$ and hence, encryption for smaller ring dimensions are more suitable for CPU platforms. Re-encryption runtime for BV-PRE scheme increases linearly with ring dimensions but still does not grow more than twice as observed on CPU implementations. Comparing our results with Table 6 [22] for re-encryption runtimes, we get a performance improvement by a factor of 39x to 228x. Similarly, we get a peak throughput of 6.3 Mbps for the BV-PRE re-encryption procedure from GPU2.

For the Ring-GSW PRE scheme, decryption runtimes are slightly higher than that of the BV PRE scheme because of relatively large modulus bit lengths. Further, the decryption runtimes vary very little and can be treated as constant for all practical purposes. Re-encryption runtimes are drastically higher and consequently, throughput is reduced when compared to BV PRE re-encryption runtimes and throughput. This is due to the fact that the relinearization proce-

Table 3. Experimental runtime performance of encryption, decryption, and re-encryption operation for ring dimension n at $r = 1$, $p = 5$, and 128-bits of security. Evaluation data reported for GPU1.

PRE scheme	Parameters		Runtime			Throughput		
	n	b	Enc (ms)	Dec (ms)	ReEnc (ms)	Enc (Kbps)	Dec (Kbps)	ReEnc (Kbps)
BV-PRE	512	18	2.14	0.46	0.29	239.25	1113.04	1765.51
	1024	18	2.18	0.52	0.45	469.72	1969.23	2275.55
	2048	19	2.21	0.55	0.85	926.69	3723.63	2409.41
	4096	19	2.28	0.66	1.26	1796.49	6206.06	3250.79
	8192	20	2.76	1.18	2.76	2968.11	6942.37	2968.11
Ring-GSW-PRE	1024	25	3.92	0.6	39.75	261.22	1706.66	25.76
	2048	26	5.34	0.55	61.18	383.52	3723.63	33.47
	4096	27	9.85	0.63	118.86	415.83	6501.58	34.46
	8192	28	19.89	1.08	259.02	411.86	7585.18	31.62

Table 4. Experimental runtime performance of encryption, decryption, and re-encryption operation for ring dimension n at $r = 1$, $p = 5$, and 128-bits of security. Evaluation data reported for GPU2.

PRE scheme	Parameters		Runtime			Throughput		
	n	b	Enc (ms)	Dec (ms)	ReEnc (ms)	Enc (Kbps)	Dec (Kbps)	ReEnc (Kbps)
BV-PRE	512	18	4.16	0.47	0.31	123.07	1089.36	1651.61
	1024	18	4.04	0.57	0.4	253.46	1796.49	2560
	2048	19	4.24	0.61	0.69	483.02	3357.37	2968.11
	4096	19	4.3	0.63	0.72	952.55	6501.58	5688.88
	8192	20	4.99	0.68	1.3	1641.68	12047.05	6301.53
Ring-GSW-PRE	1024	25	4.75	0.62	35.46	215.57	1651.61	28.87
	2048	26	5.22	0.67	49.5	392.33	3056.71	41.38
	4096	27	7.98	0.71	87.24	513.28	5769.01	46.95
	8192	28	12.2	0.71	166.67	671.47	11538.03	49.15

ture is performed over multiple rows of the ciphertext matrix. However, when compared to CPU implementations we still get performance improvements of 3.5x to 11x. Runtimes of both BV PRE and Ring-GSW PRE schemes can be further brought down by considering a larger relinearization window but are accompanied by larger noise growth and error bounds.

9 Conclusion and Future Work

In this work, we explored GPU acceleration of BV PRE and Ring-GSW PRE schemes and showed that GPUs are indeed capable of improving performance by more than an order of magnitude. Moreover, from our experiments, we found that GPUs are more effective in working with larger ring dimensions. We presented several lower level optimizations and parallel NTT implementations tailored specifically for GPU platforms which can be further extended to accelerate other FHE schemes and applications.

In this direction, we would like to support the acceleration of other compute intensive tasks such as bootstrapping LWE FHE schemes, machine learning on encrypted data and other similar privacy concerning applications.

References

1. Albrecht, M.R., Cid, C., Faugere, J.C., Fitzpatrick, R., Perret, L.: On the complexity of the BKW algorithm on LWE. *Des. Codes Cryptogr.* **74**(2), 325–354 (2015)
2. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **9**(1), 1–30 (2006)
3. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: Nyberg, K. (ed.) *EUROCRYPT 1998*. LNCS, vol. 1403, pp. 127–144. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0054122>
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory (TOCT)* **6**, 13 (2014)
5. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) *CRYPTO 2011*. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_29
6. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.* **43**(2), 831–871 (2014)
7. Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 185–194. ACM (2007)
8. Dai, W., Doröz, Y., Sunar, B.: Accelerating NTRU based homomorphic encryption using GPUs. In: *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6. IEEE (2014)
9. Dai, W., Sunar, B.: cuHE: a homomorphic encryption accelerator library. In: Pasalic, E., Knudsen, L.R. (eds.) *BalkanCryptSec 2015*. LNCS, vol. 9540, pp. 169–186. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29172-7_11

10. Dhem, J.-F., Quisquater, J.-J.: Recent results on modular multiplications for smart cards. In: Quisquater, J.-J., Schneier, B. (eds.) *CARDIS 1998*. LNCS, vol. 1820, pp. 336–352. Springer, Heidelberg (2000). https://doi.org/10.1007/10721064_31
11. Doröz, Y., Hu, Y., Sunar, B.: Homomorphic AES evaluation using NTRU. *IACR Cryptology ePrint Archive*, vol. 2014, p. 39 (2014)
12. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) *CRYPTO 2012*. LNCS, vol. 7417, pp. 850–867. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_49
13. Gentry, C., Sahai, A., Waters, B.: homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) *CRYPTO 2013*. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5
14. Gentry, C., et al.: Fully homomorphic encryption using ideal lattices. In: *STOC*, vol. 9, pp. 169–178 (2009)
15. Halevi, S., Shoup, V.: Algorithms in HELib. In: Garay, J.A., Gennaro, R. (eds.) *CRYPTO 2014*. LNCS, vol. 8616, pp. 554–571. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_31
16. Khedr, A., Gulak, G., Vaikuntanathan, V.: SHIELD: scalable homomorphic implementation of encrypted data-classifiers. *IEEE Trans. Comput.* **65**(9), 2848–2858 (2016)
17. Khurana, H., Heo, J., Pant, M.: From proxy encryption primitives to a deployable secure-mailing-list solution. In: Ning, P., Qing, S., Li, N. (eds.) *ICICS 2006*. LNCS, vol. 4307, pp. 260–281. Springer, Heidelberg (2006). https://doi.org/10.1007/11935308_19
18. Laine, K., Lauter, K.: Key recovery for LWE in polynomial time (2015)
19. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) *CT-RSA 2011*. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19074-2_21
20. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, pp. 1219–1234 (2012)
21. Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) *EUROCRYPT 2012*. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41
22. Polyakov, Y., Rohloff, K., Sahu, G., Vaikuntanathan, V.: Fast proxy re-encryption for publish/subscribe systems. *ACM Trans. Priv. Secur. (TOPS)* **20**(4), 14 (2017)
23. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. *J. ACM (JACM)* **56**(6), 34 (2009)
24. Taban, G., Cárdenas, A.A., Gligor, V.D.: Towards a secure and interoperable DRM architecture. In: *Proceedings of the ACM Workshop on Digital Rights Management*, pp. 69–78. ACM (2006)

Author Index

- Alupotha, Jayamine 430
Anand, P. Mohan 127
Ateniese, Giuseppe 323
- Baghery, Karim 453
Balli, Fatih 23
Banik, Subhadeep 23
Bellini, Emanuele 570
Bosk, Daniel 147
Bouget, Simon 147
Boyen, Xavier 430
Brannigan, Séamus 593
Buchegger, Sonja 147
- Caforio, Andrea 23
Campos, Fabio 526
Charan, P. V. Sai 127
Clarisse, Rémi 280
Conti, Mauro 344
- Dallmeier, Fynn 211
David, Bernardo 462
de Saint Guilhem, Cyprien Delpech 235
Di Luzio, Adriano 323
Dowsley, Rafael 462
Drees, Jan P. 211
Duquesne, Sylvain 280
Dutta, Ratna 387
- El Housni, Youssef 259
ElSheikh, Muhammad 485
- Foo, Ernest 430
Francati, Danilo 323
- Gaborit, Philippe 570
Gangwal, Ankit 344
Gelernter, Nethanel 43
Gellert, Kai 211
Greuet, Aurélien 549
Guillevic, Aurore 259
Güneysu, Tim 299
Gupta, Debayan 167
- Handirk, Tobias 211
Harikrishnan, T. S. 167
Hasikos, Alexandros 570
Herzberg, Amir 43
Hidano, Seira 65
Hiji, Masahiro 65
Huszt, Andrea 188
- Jager, Tibor 211
Jellema, Lars 526
Jourenko, Maxim 365
- Kales, Daniel 3
Karakoç, Ferhat 409
Khalid, Ayesha 593
Kiyomoto, Shinsaku 65
Klauke, Jonas 211
Kleijnung, Thorsten 299
Kumar, Saurabh 85
Küpçü, Alptekin 409
- Lain, Gianluca 344
Larangeira, Mario 365
Lemmen, Mauk 526
- Mantel, Heiko 505
Mateu, Victor 570
Meyuhas, Bar 43
Mishra, Debadatta 85
Montoya, Simon 549
Müller, Lars 526
- Nachtigall, Simon 211
Narisada, Shintaro 65
- O'Neill, Maire 593
Oláh, Norbert 188
Orsini, Emmanuela 235
- Pal, Tapas 387
Panda, Biswabandan 85
Petit, Christophe 235
Piazzetta, Samuele Giuliano 344

- Picazo-Sanchez, Pablo 107
Pindado, Zaira 453
Priplata, Christine 299
- Rafferty, Ciara 593
Ràfols, Carla 453
Renault, Guénaël 549
Renzelmann, Timo 211
Richter-Brockmann, Jan 299
Rohloff, Kurt 613
Rooparaghunath, Ruthu Hulikal 167
- Sabelfeld, Andrei 107
Sahu, Gyana 613
Sanders, Olivier 280
Sasaki, Shoichiro 65
Scheidel, Lukas 505
Schneider, Gerardo 107
Schneider, Thomas 505
Shukla, Sandeep K. 85, 127
- Smart, Nigel P. 235
Sprenkels, Amber 526
Stahlke, Colin 299
Suganuma, Takuo 65
- Tanaka, Keisuke 365
- Uchibayashi, Toshihiro 65
- Viguiet, Benoit 526
- Weber, Alexandra 505
Weinert, Christian 505
Weißmantel, Tim 505
Wloka, Jonas 299
Wolf, Rudi 211
- Youssef, Amr M. 485
- Zaverucha, Greg 3