

Carmit Hazay  
Martijn Stam (Eds.)

LNCS 14005

# Advances in Cryptology – EUROCRYPT 2023

42nd Annual International Conference on the Theory  
and Applications of Cryptographic Techniques  
Lyon, France, April 23–27, 2023, Proceedings, Part II

2  
Part II



 Springer

MOREMEDIA 

# Lecture Notes in Computer Science

14005

## Founding Editors

Gerhard Goos  
Juris Hartmanis

## Editorial Board Members

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

Wen Gao, *Peking University, Beijing, China*

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

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

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

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


Carmit Hazay · Martijn Stam  
Editors


# Advances in Cryptology – EUROCRYPT 2023

42nd Annual International Conference on the Theory  
and Applications of Cryptographic Techniques  
Lyon, France, April 23–27, 2023  
Proceedings, Part II

 Springer

*Editors*

Carmit Hazay   
Bar-Ilan University  
Ramat Gan, Israel

Martijn Stam   
Simula UiB  
Bergen, Norway

ISSN 0302-9743

ISSN 1611-3349 (electronic)

Lecture Notes in Computer Science

ISBN 978-3-031-30616-7

ISBN 978-3-031-30617-4 (eBook)

<https://doi.org/10.1007/978-3-031-30617-4>

© International Association for Cryptologic Research 2023

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

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

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

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

# Preface

The 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Eurocrypt 2023, was held in Lyon, France between April 23–27 under the auspices of the International Association for Cryptologic Research. The conference had a record number of 415 submissions, out of which 109 were accepted.

Preparation for the academic aspects of the conference started in earnest well over a year ago, with the selection of a program committee, consisting of 79 regular members and six area chairs. The area chairs played an important part in enabling a high-quality review process; their role was expanded considerably from last year and, for the first time, properly formalized. Each area chair was in charge of moderating the discussions of the papers assigned under their area, guiding PC members and reviewers to consensus where possible, and helping us in making final decisions. We created six areas and assigned the following area chairs: Ran Canetti for Theoretical Foundations; Rosario Gennaro for Public Key Primitives with Advanced Functionalities; Tibor Jager for Classic Public Key Cryptography; Marc Joye for Secure and Efficient Implementation, Cryptographic Engineering, and Real-World Cryptography; Gregor Leander for Symmetric Cryptology; and finally Arpita Patra for Multi-party Computation and Zero-Knowledge.

Prior to the submission deadline, PC members were introduced to the reviewing process; for this purpose we created a slide deck that explained what we expected from everyone involved in the process and how PC members could use the reviewing system (HotCRP) used by us. An important aspect of the reviewing process is the reviewing form, which we modified based on the Crypto'22 form as designed by Yevgeniy Dodis and Tom Shrimpton. As is customary for IACR general conferences, the reviewing process was two-sided anonymous.

Out of the 415 submissions, four were desk rejected due to violations of the Call for Papers (non-anonymous submission or significant deviations from the submission format). For the remaining submissions, the review process proceeded in two stages. In the first stage, every paper was reviewed by at least three reviewers. For 109 papers a clear, negative consensus emerged and an early reject decision was reached and communicated to the authors on the 8th of December 2022. This initial phase of early rejections allowed the program committee to concentrate on the delicate task of selecting a program amongst the more promising submissions, while simultaneously offering the authors of the rejected papers the opportunity to take advantage of the early, full feedback to improve their work for a future occasion.

The remaining 302 papers progressed to an interactive discussion phase, which was open for two weeks (ending slightly before the Christmas break). During this period, the authors had access to their reviews (apart from some PC only fields) and were asked to address questions and requests for clarifications explicitly formulated in the reviews. It gave authors and reviewers the opportunity to communicate directly (yet anonymously) with each other during several rounds of interaction. For some papers, the multiple rounds helped in clarifying both the reviewers' questions and the authors' responses.

For a smaller subset of papers, a second interactive discussion phase took place in the beginning of January allowing authors to respond to new, relevant insights by the PC. Eventually, 109 papers were selected for the program.

The best paper award was granted to the paper “An Efficient Key Recovery Attack on SIDH” by Wouter Castryck and Thomas Decru for presenting the first efficient key recovery attack against the Supersingular Isogeny Diffie-Hellman (SIDH) problem. Two further, related papers were invited to the Journal of Cryptology: “Breaking SIDH in Polynomial Time” by Damien Robert and “A Direct Key Recovery Attack on SIDH” by Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope and Benjamin Wesolowski.

Accepted papers written exclusively by researchers who were within four years of PhD graduation at the time of submission were eligible for the Early Career Best Paper Award. There were a number of strong candidates and the paper “Worst-Case Subexponential Attacks on PRGs of Constant Degree or Constant Locality” by Akın Ünäl was awarded this honor.

The program further included two invited talks: Guy Rothblum opened the program with his talk on “Indistinguishable Predictions and Multi-group Fair Learning” (an extended abstract of his talk appears in these proceedings) and later during the conference Vadim Lyubashevsky gave a talk on “Lattice Cryptography: What Happened and What’s Next”.

First and foremost, we would like to thank Kevin McCurley and Kay McKelly for their tireless efforts in the background, making the whole process so much smoother for us to run. Thanks also to our previous co-chairs Orr Dunkelman, Stefan Dziembowski, Yevgeniy Dodis, Thomas Shrimpton, Shweta Agrawal and Dongdai Lin for sharing the lessons they learned and allowing us to build on their foundations. We thank Guy and Vadim for accepting to give two excellent invited talks. Of course, no program can be selected without submissions, so we thank both the authors of accepted papers, as well as those whose papers did not make it (we sincerely hope that, notwithstanding the disappointing outcome, you found the reviews and interaction constructive). The reviewing was led by our PC members, who often engaged expert subreviewers to write high-quality, insightful reviews and engage directly in the discussions, and we are grateful to both our PC members and the subreviewers. As the IACR’s general conferences grow from year to year, a very special thank you to our area chairs, our job would frankly not have been possible without Ran, Rosario, Tibor, Marc, Gregor, and Arpita’s tireless efforts leading the individual papers’ discussions. And, last but not least, we would like to thank the general chairs: Damien Stehlé, Alain Passelègue, and Benjamin Wesolowski who worked very hard to make this conference happen.

April 2023

Carmit Hazay  
Martijn Stam

# Organization

## General Co-chairs

Damien Stehlé	ENS de Lyon and Institut Universitaire de France, France
Alain Passelègue	Inria, France
Benjamin Wesolowski	CNRS and ENS de Lyon, France

## Program Co-chairs

Carmit Hazay	Bar-Ilan University, Israel
Martijn Stam	Simula UiB, Norway

## Area Chairs

Ran Canetti <i>(for Theoretical Foundations)</i>	Boston University, USA
Rosario Gennaro <i>(for Public Key Primitives with Advanced Functionalities)</i>	Protocol Labs and CUNY, USA
Tibor Jager <i>(for Classic Public Key Cryptography)</i>	University of Wuppertal, Germany
Marc Joye <i>(for Secure and Efficient Implementation, Cryptographic Engineering, and Real-World Cryptography)</i>	Zama, France
Gregor Leander <i>(for Symmetric Cryptology)</i>	Ruhr-Universität Bochum, Germany
Arpita Patra <i>(for Multi-party Computation and Zero-Knowledge)</i>	Google and IISc Bangalore, India



**Program Committee**

Masayuki Abe	NTT Social Informatics Laboratories and Kyoto University, Japan
Adi Akavia	University of Haifa, Israel
Prabhanjan Ananth	UC Santa Barbara, USA
Gilad Asharov	Bar-Ilan University, Israel
Marshall Ball	New York University, USA
Christof Beierle	Ruhr University Bochum, Germany
Mihir Bellare	UC San Diego, USA
Tim Beyne	KU Leuven, Belgium
Andrej Bogdanov	Chinese University of Hong Kong, China
Xavier Bonnetain	Inria, France
Joppe Bos	NXP Semiconductors, Belgium
Chris Brzuska	Aalto University, Finland
Ignacio Cascudo	IMDEA Software Institute, Spain
Nishanth Chandran	Microsoft Research India, India
Chitchanok Chuengsatiansup	The University of Melbourne, Australia
Michele Ciampi	The University of Edinburgh, UK
Ran Cohen	Reichman University, Israel
Jean-Sébastien Coron	University of Luxembourg, Luxembourg
Bernardo David	IT University of Copenhagen, Denmark
Christoph Dobraunig	Intel Labs, Intel Corporation, Hillsboro, USA
Léo Ducas	CWI Amsterdam and Leiden University, Netherlands
Maria Eichlseder	Graz University of Technology, Austria
Pooya Farshim	IOHK and Durham University, UK
Serge Fehr	CWI Amsterdam and Leiden University, Netherlands
Dario Fiore	IMDEA Software Institute, Spain
Pierre-Alain Fouque	Université Rennes 1 and Institut Universitaire de France, France
Steven Galbraith	University of Auckland, New Zealand
Chaya Ganesh	IISc Bangalore, India
Si Gao	Huawei Technologies Co., Ltd., China
Daniel Genkin	GeorgiaTech, USA
Craig Gentry	TripleBlind, USA
Benedikt Gierlichs	KU Leuven, Belgium
Rishab Goyal	UW-Madison, USA
Vipul Goyal	NTT Research and CMU, USA
Viet Tung Hoang	Florida State University, USA
Andreas Hülsing	Eindhoven University of Technology, Netherlands

Antoine Joux	CISPA, Helmholtz Center for Cybersecurity, Germany
Karen Klein	ETH Zurich, Switzerland
Markulf Kohlweiss	University of Edinburgh and IOHK, UK
Jooyoung Lee	KAIST, Korea
Gaëtan Leurent	Inria, France
Shengli Liu	Shanghai Jiao Tong University, China
Yunwen Liu	Cryptape Technology Co., Ltd., China
Stefan Lucks	Bauhaus-Universität Weimar, Germany
Hemanta Maji	Purdue, USA
Alexander May	Ruhr University Bochum, Germany
Nele Mentens	Leiden University, Netherlands and KU Leuven, Belgium
Tal Moran	Reichman University, Israel
Michael Naehrig	Microsoft Research, USA
Ngoc Khanh Nguyen	EPFL, Switzerland
Emmanuela Orsini	Bocconi University, Italy and KU Leuven, Belgium
Jiaxin Pan	NTNU, Norway
Omkant Pandey	Stony Brook University, USA
Anat Paskin-Cherniavsky	Ariel University, Israel
Chris Peikert	University of Michigan and Algorand, Inc., USA
Léo Perrin	Inria, France
Giuseppe Persiano	Università di Salerno, Italy
Thomas Peters	UCLouvain, Belgium
Christophe Petit	Université libre de Bruxelles, Belgium and University of Birmingham, UK
Krzysztof Pietrzak	ISTA, Austria
Bertram Poettering	IBM Research Europe – Zurich, Switzerland
Bart Preneel	KU Leuven, Belgium
Divya Ravi	Aarhus University, Denmark
Christian Rechberger	TU Graz, Austria
Ron Rothblum	Technion, Israel
Carla Ràfols	Universitat Pompeu Fabra, Spain
Paul Rösler	FAU Erlangen-Nürnberg, Germany
Yu Sasaki	NTT Social Informatics Laboratories, NIST Associate, Japan
Dominique Schröder	FAU Erlangen-Nürnberg, Germany
Omri Shmueli	Tel Aviv University, Israel
Janno Siim	Simula UiB, Norway
Daniel Slamanig	AIT Austrian Institute of Technology, Austria
Yifan Song	Tsinghua University, China

Qiang Tang	The University of Sydney, Australia
Serge Vaudenay	EPFL, Switzerland
Fernando Virdia	Intel Labs, Switzerland
Meiqin Wang	Shandong University, China
Mor Weiss	Bar-Ilan University, Israel
David Wu	UT Austin, USA

## Additional Reviewers

Behzad Abdolmaleki	Katharina Boudgoust
Damiano Abram	Christina Boura
Hamza Abusalah	Zvika Brakerski
Leo Ackermann	Lennart Braun
Amit Agarwal	Marek Broll
Ghous Amjad	Ileana Buhan
Benny Applebaum	Matteo Campanelli
Gal Arnon	Federico Canale
Thomas Attema	Anne Canteaut
Benedikt Auerbach	Gaëtan Cassiers
Lukas Aumayr	Wouter Castryck
Gennaro Avitabile	Pyrros Chaidos
Melissa Azouaoui	André Chailloux
Saikrishna Badrinarayanan	T.-H. Hubert Chan
Karim Baghery	Anirudh Chandramouli
Kunpeng Bai	Rohit Chatterjee
Shi Bai	Hao Chen
David Balbás	Long Chen
Manuel Barbosa	Mingjie Chen
Khashayar Barooti	Yanbo Chen
James Bartusek	Yanlin Chen
Andrea Basso	Yilei Chen
Balthazar Bauer	Yu Long Chen
Carsten Baum	Wei Cheng
Michiel van Beirendonck	Céline Chevalier
Josh Benaloh	James Chiang
Fabrice Benhamouda	Wonhee Cho
Ward Beullens	Wonseok Choi
Amit Singh Bhati	Wutichai Chongchitmate
Ritam Bhaumik	Hien Chu
Alexander Bienstock	Valerio Cini
Alexander Block	Christine Cloostermans
Jonathan Bootle	Andrea Coladangelo
Cecilia Boschini	Daniel Collins

Sandro Coretti-Drayton  
Craig Costello  
Elizabeth Crites  
Miguel Cueto Noval  
Jan-Pieter D’Anvers  
Sourav Das  
Alex Davidson  
Gabrielle De Micheli  
Cyprien Delpech de Saint Guilhem  
Patrick Derbez  
Lalita Devadas  
Siemen Dhooghe  
Jesus Diaz  
Khue Do  
Jelle Don  
Rafael Dowsley  
Avijit Dutta  
Sébastien Duval  
Christoph Egger  
Tariq Elahi  
Lynn Engelberts  
Felix Engelmann  
Muhammed F. Esgin  
Thomas Espitau  
Andre Esser  
Simona Etinski  
Prastudy Fauzi  
Patrick Felke  
Hanwen Feng  
Rex Fernando  
Tako Boris Fouotsa  
Danilo Francati  
Sapir Freizeit  
Paul Frixons  
Rachit Garg  
Sanjam Garg  
Aymeric Genêt  
Marios Georgiou  
Satrajit Ghosh  
Niv Gilboa  
Valerie Gilchrist  
Emanuele Giunta  
Aarushi Goel  
Eli Goldin  
Junqing Gong  
Alonso González  
Lorenzo Grassi  
Jiaxin Guan  
Zichen Gui  
Aurore Guillevic  
Aditya Gulati  
Aldo Gunsing  
Chun Guo  
Divya Gupta  
Felix Günther  
Hosein Hadipour  
Mohammad Hajiabadi  
Shai Halevi  
Peter Hall  
Shuai Han  
Patrick Harasser  
David Heath  
Lena Heimberger  
Alexandra Henzinger  
Julia Hesse  
Minki Hhan  
Dennis Hofheinz  
Maya-Iggy van Hoof  
Sam Hopkins  
Akinori Hosoyamada  
Kristina Hostáková  
Martha Norberg Hovd  
Yu-Hsuan Huang  
Loïs Huguenin-Dumittan  
Kathrin Hövelmanns  
Yuval Ishai  
Muhammad Ishaq  
Tetsu Iwata  
Michael John Jacobson, Jr.  
Aayush Jain  
Samuel Jaques  
Jinhyuck Jeong  
Corentin Jeudy  
Ashwin Jha  
Mingming Jiang  
Zhengzhong Jin  
Thomas Johansson  
David Joseph  
Daniel Jost  
Fatih Kaleoglu

Novak Kaluderovic  
Chethan Kamath  
Shuichi Katsumata  
Marcel Keller  
John Kelsey  
Erin Kenney  
Hamidreza Khorasgani  
Hamidreza Khoshakhlagh  
Seongkwang Kim  
Elena Kirshanova  
Fuyuki Kitagawa  
Bor de Kock  
Konrad Kohbrok  
Lisa Kohl  
Sebastian Kolby  
Dimitris Kolonelos  
Ilan Komargodski  
Yashvanth Kondi  
Venkata Koppula  
Alexis Korb  
Matthias Krause  
Hugo Krawczyk  
Toomas Krips  
Mike Kudinov  
Péter Kutas  
Thijs Laarhoven  
Yi-Fu Lai  
Baptiste Lambin  
Nathalie Lang  
Abel Laval  
Laurens Le Jeune  
Byeonghak Lee  
Changmin Lee  
Eysa Lee  
Seunghoon Lee  
Sihyun Lee  
Dominik Leichtle  
Jannis Leuther  
Shai Levin  
Chaoyun Li  
Yanan Li  
Yiming Li  
Xiao Liang  
Jyun-Jie Liao  
Benoît Libert  
Wei-Kai Lin  
Yao-Ting Lin  
Helger Lipmaa  
Eik List  
Fukang Liu  
Jiahui Liu  
Qipeng Liu  
Xiangyu Liu  
Chen-Da Liu-Zhang  
Satya Lokam  
Alex Lombardi  
Patrick Longa  
George Lu  
Jinyu Lu  
Xianhui Lu  
Yuan Lu  
Zhenliang Lu  
Ji Luo  
You Lyu  
Reinhard Lüftenegger  
Urmila Mahadev  
Mohammad Mahmoody  
Mohammad Mahzoun  
Christian Majenz  
Nikolaos Makriyannis  
Varun Maram  
Laurane Marco  
Ange Martinelli  
Daniel Masny  
Noam Mazor  
Matthias Meijers  
Fredrik Meisingseth  
Florian Mendel  
Bart Mennink  
Simon-Philipp Merz  
Tony Metger  
Pierre Meyer  
Brice Minaud  
Kazuhiko Minematsu  
Victor Mollimard  
Tomoyuki Morimae  
Nicky Mouha  
Tamer Mour  
Marcel Nageler  
Mridul Nandi

María Naya-Plasencia  
Patrick Neumann  
Hai Nguyen  
Ky Nguyen  
Phong Q. Nguyen  
Ryo Nishimaki  
Olga Nissenbaum  
Anca Nitulescu  
Ariel Nof  
Julian Nowakowski  
Adam O'Neill  
Sai Lakshmi Bhavana Obbattu  
Miyako Ohkubo  
Eran Omri  
Claudio Orlandi  
Michele Orrù  
Elisabeth Oswald  
Omer Paneth  
Guillermo Pascual-Perez  
Kenneth G. Paterson  
Sikhar Patranabis  
Alice Pellet-Mary  
Maxime Plancon  
Antigoni Polychroniadou  
Alexander Poremba  
Bernardo Portela  
Eamonn Postlethwaite  
Emmanuel Prouff  
Kirthivaasan Puniamurthy  
Octavio Pérez Kempner  
Luowen Qian  
Tian Qiu  
Willy Quach  
Håvard Raddum  
Srinivasan Raghuraman  
Justin Raizes  
Sebastian Ramacher  
Hugues Randriambololona  
Shahram Rasoolzadeh  
Simon Rastikian  
Joost Renes  
Nicolas Resch  
Alfredo Rial Duran  
Doreen Riepel  
Silvia Ritsch  
Melissa Rossi  
Mike Rosulek  
Yann Rotella  
Lawrence Roy  
Roozbeh Sarenche  
Amirreza Sarencheh  
Pratik Sarkar  
Arish Sateesan  
Christian Schaffner  
Carl Richard Theodor Schneider  
Markus Schofnegger  
Peter Scholl  
André Schrottenloher  
Gregor Seiler  
Sruthi Sekar  
Nicolas Sendrier  
Meghna Sengupta  
Jinrui Sha  
Akash Shah  
Siamak Shahandashti  
Moni Shahar  
Shahed Sharif  
Laura Shea  
Abhi Shelat  
Yaobin Shen  
Sina Shiehian  
Jad Silbak  
Alice Silverberg  
Luisa Siniscalchi  
Tomer Solomon  
Karl Southern  
Nicholas Spooner  
Sriram Sridhar  
Srivatsan Sridhar  
Akshayaram Srinivasan  
François-Xavier Standaert  
Uri Stemmer  
Lukas Stennes  
Patrick Steuer  
Christoph Striecks  
Patrick Struck  
Chao Sun  
Erkan Tairi  
Akira Takahashi  
Abdullah Talayhan

Titouan Tanguy	Weiqliang Wen
Stefano Tessaro	Benjamin Wesolowski
Emmanuel Thomé	Daniel Wicks
Sri AravindaKrishnan Thyagarajan	Wessel van Woerden
Yan Bo Ti	Ke Wu
Mehdi Tibouchi	Keita Xagawa
Tyge Tiessen	Hanshen Xiao
Bénédict Tran	Jiayu Xu
Andreas Trügler	Yingfei Yan
Daniel Tschudi	Xiuyu Ye
Aleksei Udovenko	Kevin Yeo
Jonathan Ullman	Eylon Yogev
Dominique Unruh	Albert Yu
Vinod Vaikuntanathan	Aaram Yun
Daniele Venturi	Alexandros Zacharakis
Michiel Verbauwhede	Thomas Zacharias
Javier Verbel	Michal Zajac
Gilles Villard	Greg Zaverucha
Mikhail Volkhov	Runzhi Zeng
Satyanarayana Vusirikala	Cong Zhang
Benedikt Wagner	Lei Zhang
Roman Walch	Ren Zhang
Hendrik Waldner	Xinrui Zhang
Alexandre Wallet	Yuqing Zhao
Michael Walter	Yu Zhou
Mingyuan Wang	Dionysis Zindros
Yuyu Wang	Giorgos Zirdelis
Florian Weber	Lukas Zobernig
Hoeteck Wee	Arne Tobias Ødegaard
Puwen Wei	Morten Øyegarden
Charlotte Weitkaemper	

## Sponsoring Institutions

- Platinum Sponsor: Université Rennes 1 and PEPR Quantique, Zama
- Gold Sponsor: Apple, Cryptolab, ENS de Lyon, ENS PSL, Huawei, Sandbox AQ, Thales, TII
- Silver Sponsor: Algorand Foundation, ANSSI, AWS, PQShield
- Bronze Sponsor: Cosmian, CryptoExperts, CryptoNext Security, IBM, Idemia, Inria, LIP

## Contents – Part II

### Multi-party Computation

New Ways to Garble Arithmetic Circuits .....	3
<i>Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu</i>	
Actively Secure Half-Gates with Minimum Overhead Under Duplex Networks .....	35
<i>Hongrui Cui, Xiao Wang, Kang Yang, and Yu Yu</i>	
Black-Box Reusable NISC with Random Oracles .....	68
<i>Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan</i>	
Maliciously-Secure MrNISC in the Plain Model .....	98
<i>Rex Fernando, Aayush Jain, and Ilan Komargodski</i>	
Minimizing Setup in Broadcast-Optimal Two Round MPC .....	129
<i>Ivan Damgård, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov</i>	
Sublinear-Communication Secure Multiparty Computation Does Not Require FHE .....	159
<i>Elette Boyle, Geoffroy Couteau, and Pierre Meyer</i>	
Actively Secure Arithmetic Computation and VOLE with Constant Computational Overhead .....	190
<i>Benny Applebaum and Niv Konstantini</i>	
SUPERPACK: Dishonest Majority MPC with Constant Online Communication .....	220
<i>Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, Yifan Song, and Chenkai Weng</i>	
Detect, Pack and Batch: Perfectly-Secure MPC with Linear Communication and Constant Expected Time .....	251
<i>Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra</i>	
An Incremental PoSW for General Weight Distributions .....	282
<i>Hamza Abusalah and Valerio Cini</i>	



**(Zero-Knowledge) Proofs**

<b>Witness-Succinct Universally-Composable SNARKs</b> .....	315
<i>Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi</i>	
<b>Speed-Stacking: Fast Sublinear Zero-Knowledge Proofs for Disjunctions</b> .....	347
<i>Aarushi Goel, Mathias Hall-Andersen, Gabriel Kaptchuk, and Nicholas Spooner</i>	
<b>Proof-Carrying Data from Arithmetized Random Oracles</b> .....	379
<i>Megan Chen, Alessandro Chiesa, Tom Gur, Jack O’Connor, and Nicholas Spooner</i>	
<b>Supersingular Curves You Can Trust</b> .....	405
<i>Andrea Basso, Giulio Codogni, Deirdre Connolly, Luca De Feo, Tako Boris Fouotsa, Guido Maria Lido, Travis Morrison, Lorenz Panny, Sikhar Patranabis, and Benjamin Wesolowski</i>	
<b>On Valiant’s Conjecture: Impossibility of Incrementally Verifiable Computation from Random Oracles</b> .....	438
<i>Mathias Hall-Andersen and Jesper Buus Nielsen</i>	
<b>SNARGs and PPAD Hardness from the Decisional Diffie-Hellman Assumption</b> .....	470
<i>Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan</i>	
<b>HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates</b> .....	499
<i>Binyi Chen, Benedikt Büinz, Dan Boneh, and Zhenfei Zhang</i>	
<b>Spartan and Bulletproofs are Simulation-Extractable (for Free!)</b> .....	531
<i>Quang Dao and Paul Grubbs</i>	
<b>Complete Characterization of Broadcast and Pseudo-signatures from Correlations</b> .....	563
<i>Varun Narayanan, Vinod M. Prabhakaran, Neha Sangwan, and Shun Watanabe</i>	
<b>Privacy-Preserving Blueprints</b> .....	594
<i>Markulf Kohlweiss, Anna Lysyanskaya, and An Nguyen</i>	
<b>Author Index</b> .....	627

# **Multi-party Computation**



# New Ways to Garble Arithmetic Circuits

Marshall Ball<sup>1</sup>, Hanjun Li<sup>2(✉)</sup>, Huijia Lin<sup>2</sup>, and Tianren Liu<sup>3</sup>

<sup>1</sup> New York University, New York, USA  
marshall@cs.nyu.edu

<sup>2</sup> University of Washington, Seattle, USA  
{hanjul,rachel}@cs.washington.edu

<sup>3</sup> Peking University, Beijing, China  
trl@pku.edu.cn

**Abstract.** The beautiful work of Applebaum, Ishai, and Kushilevitz [FOCS’11] initiated the study of arithmetic variants of Yao’s garbled circuits. An arithmetic garbling scheme is an efficient transformation that converts an arithmetic circuit  $C : \mathcal{R}^n \rightarrow \mathcal{R}^m$  over a ring  $\mathcal{R}$  into a garbled circuit  $\widehat{C}$  and  $n$  affine functions  $L_i$  for  $i \in [n]$ , such that  $\widehat{C}$  and  $L_i(x_i)$  reveals only the output  $C(x)$  and no other information of  $x$ . AIK presented the first arithmetic garbling scheme supporting computation over integers from a bounded (possibly exponentially large) range, based on Learning With Errors (LWE). In contrast, converting  $C$  into a Boolean circuit and applying Yao’s garbled circuit treats the inputs as bit strings instead of ring elements, and hence is not “arithmetic”.

In this work, we present new ways to garble arithmetic circuits, which improve the state-of-the-art on efficiency, modularity, and functionality. To measure efficiency, we define the rate of a garbling scheme as the maximal ratio between the bit-length of the garbled circuit  $|\widehat{C}|$  and that of the computation tableau  $|C|\ell$  in the clear, where  $\ell$  is the bit length of wire values (e.g., Yao’s garbled circuit has rate  $O(\lambda)$ ).

- We present the first *constant-rate* arithmetic garbled circuit for computation over large integers based on the Decisional Compositeness Residuosity (DCR) assumption, significantly improving the efficiency of the schemes of Applebaum, Ishai, and Kushilevitz.
- We construct an arithmetic garbling scheme for modular computation over  $\mathcal{R} = \mathbb{Z}_p$  for any integer modulus  $p$ , based on either DCR or LWE. The DCR-based instantiation achieves rate  $O(\lambda)$  for large  $p$ . Furthermore, our construction is modular and makes black-box use of the underlying ring and a simple *key extension* gadget.
- We describe a variant of the first scheme supporting arithmetic circuits over bounded integers that are augmented with Boolean computation (e.g., truncation of an integer value, and comparison between two values), while keeping the *constant rate* when garbling the arithmetic part.

To the best of our knowledge, constant-rate (Boolean or arithmetic) garbling was only achieved before using the powerful primitive of indistinguishability obfuscation, or for restricted circuits with small depth.

## 1 Introduction

Garbled circuits, introduced by Yao [18], enable a “Garbler” to efficiently transform a Boolean circuit  $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$  into a *garbled circuit*  $\widehat{C}$  and a pair of keys  $\mathbf{k}_0^i, \mathbf{k}_1^i$  for every input bit. In particular, the input keys are short, of length polynomial in the security parameter only, independent of the complexity of the circuit. An input  $x \in \{0, 1\}^n$  to the circuit can be encoded by choosing the right keys corresponding to each input bit  $\mathbf{L}^x = \{\mathbf{k}_{x_i}^i\}_{i \in [n]}$ , referred to as the input *labels*. The garbled circuit and input labels  $(\widehat{C}, \mathbf{L}^x)$  together reveal the output of the computation  $y = C(x)$ , and hide all other information of  $x$ . Yao’s seminal result [18] constructed garbled circuit using Pseudo-Random Generators (PRGs), which in turn can be based on one-way functions. Since its conception, garbled circuits has found a wide range of applications, and is recognized as one of the most fundamental and useful tools in cryptography.

**The Arithmetic Setting.** While there have been remarkable optimizations and analytical improvements in the intervening years, the currently most widely applied approaches to garbling circuits still largely follow Yao’s paradigm from the 1980s<sup>1</sup>. Yao’s idea involves encrypting the truth tables of gates in the circuit, which becomes inefficient or even infeasible when the truth tables are large. A longstanding open question is designing *arithmetic garbling*, namely, variants of garbled circuits that apply naturally to arithmetic circuits without “Booleanizing” the computation, meaning bit-decomposing the inputs and intermediate values and garbling the Boolean circuit implementation of arithmetic operations. To achieve arithmetic garbling, fundamentally new techniques different from the mainstream encrypted truth-table methods must be developed.

The work of Applebaum, Ishai, and Kushilevitz (AIK) [5] initiated the study of arithmetic garbling. They first formalized the notion of *Decomposable Affine Randomized Encoding (DARE)* as follows:

ARITHMETIC GARBLING (I.E., DARE) is an efficient transformation *Garble* that converts an arithmetic circuit  $C : \mathcal{R}^n \rightarrow \mathcal{R}^m$  over a ring  $\mathcal{R}$  into a garbled circuit  $\widehat{C}$ , along with  $2n$  key vectors  $\mathbf{k}_0^i, \mathbf{k}_1^i \in \mathcal{R}^\ell$ , such that  $\widehat{C}$  together with the input labels  $\mathbf{L}^x = \{\mathbf{L}^i = \mathbf{k}_0^i x_i + \mathbf{k}_1^i\}$  computed over the ring  $\mathcal{R}$ , reveal  $C(x)$  and no additional information about  $x \in \mathcal{R}^n$ .

The main difference between arithmetic and Boolean garbling is that the input encoding procedure of the former consists of affine functions *over the ring*  $\mathcal{R}$ , and does not require the bit-representation of the inputs. There are natural information theoretic methods for garbling arithmetic formulas and branching programs

---

<sup>1</sup> There have been alternative approaches that rely on strong primitives such as a combination of fully homomorphic encryption and attribute-based encryption [9, 11, 15], or indistinguishability obfuscation [2]. These approaches however are much more complex than Yao’s garbling and less employed in applications. See Sect. 1.2 for more discussion.

over any ring  $\mathcal{R}$  [4, 13]. But garbling general (unbounded depth) arithmetic circuits is significantly more challenging. AIK proposed the first construction supporting *bounded integer computation* – namely computation over integers  $\mathcal{R} = \mathbb{Z}$  from a bounded (but possibly exponential) range  $[-B, B]$  – based on the Learning With Errors (LWE) assumption. In addition, they presented an alternative construction that generically reduce arithmetic garbled circuits to Yao’s Boolean garbled circuits, via a gadget that converts integer inputs into their bit representation using the Chinese Remainder Theorem (CRT). Though general, the CRT-based solution does not satisfy many desiderata of arithmetic garbling, in particular, it still relies on bit-decomposing the inputs and garbling the Boolean circuit implementation of arithmetic operations. So far, the AIK LWE-based construction gives the only known scheme that can garble to general arithmetic circuits without “Booleanizing” them.

## 1.1 Our Results

Despite its importance, little progress were made on arithmetic garbling in the past decade after the work of AIK. In this paper, we revisit this topic and present new ways of arithmetic garbling. Our contributions include 1) a significantly more efficient arithmetic garbling scheme for bounded integer computations, achieving constant rate, 2) the first scheme supporting modular arithmetic computation mod  $p$  that makes only *black-box* calls to the implementation of arithmetic operations, and 3) a new way of mixing arithmetic garbling with Boolean garbling. Finally, we diversify the assumptions, showing the Decisional Composite Residuosity (DCR) assumption is also sufficient, in addition to LWE.

**Part 1: Constant-Rate Garbling Scheme for Bounded Arithmetic.** To highlight our efficiency improvement for bounded integer garbling, we define the *rate* of a garbling scheme to be the maximal ratio between the bit-length of the produced garbled circuit  $|\widehat{C}|$  and input encoding, and the bit-length of the tableau of the computation in the clear  $|C|\ell$  (i.e., the bit length of merely writing down all the input and intermediate computation values). Let  $\ell$  be bit length of wire values. For a  $B$ -bounded integer computation,  $\ell = \lceil \log(2B + 1) \rceil$ .

$$\text{rate} = \max_{C, \mathbf{x}} \frac{|\widehat{C}| + |\mathbf{L}^{\mathbf{x}}|}{|C|\ell}$$

For example, the rate of Yao’s garbling for Boolean circuits is  $\frac{O((|C'| + |\mathbf{x}|)k_{\text{SKE}})}{|C'| \times (\ell=1)} = O(k_{\text{SKE}})$ , where  $k_{\text{SKE}}$  is the key length of the symmetric key encryption (or PRF) used. For arithmetic garbling, the Boolean baseline of applying Yao’s garbling on the Boolean circuit implementation of the arithmetic circuit achieves a rate of  $O(\log \ell \cdot k_{\text{SKE}})$ , when implementing integer addition/multiplication using the most asymptotically efficient algorithms of complexity  $O(\ell \log \ell)$  [12]<sup>2</sup>. The

<sup>2</sup> Note that this approach is entirely impractical for any reasonable length input due to the astronomical constants involved in fast multiplication.

**Table 1.** Comparison of Arithmetic Garbling for Bounded Integer Computation.

Garbling Scheme	Assumption	Rate	Input Label Size
Boolean Baseline	OWFs	$O(k_{\text{SKE}} \log \ell)$	$O(n\ell k_{\text{SKE}})$
AIK - CRT-based [5]	OWF	$O(k_{\text{SKE}} \log \ell)$	$O(n\ell^6 k_{\text{SKE}})$
AIK - LWE-based [5]	LWE	$O(k_{\text{LWE}})$	$\tilde{O}(n\ell k_{\text{LWE}})$
This work	DCR	$O(1 + \frac{k_{\text{DCR}}}{\ell})$	$O(n(k_{\text{DCR}} + \ell))$

CRT-based construction by AIK reduces arithmetic garbling to Yao’s Boolean garbling and achieves the same asymptotic rate  $O(\log \ell \cdot k_{\text{SKE}})$  when the circuit size is sufficient large. However, the size of the input labels is  $O(n\ell^6 k_{\text{SKE}})$  where  $n$  is the number of input elements, which is prohibitive even for relatively small range, say 10-bit, integer computation. The AIK LWE-based construction, on the other hand, has a larger rate of  $O(k_{\text{LWE}})$  where  $k_{\text{LWE}}$  is the LWE dimension, which must be larger than  $\ell^{1+\epsilon}$  for some constant  $\epsilon \in (0, 1)$ . See Table 1 for a summary.

We show that arithmetic garbling can actually be significantly more efficient than the Boolean baseline. Based on the Decisional Composite Residuosity (DCR) assumption over Paillier groups  $\mathbb{Z}_{N^{r+1}}^*$  for  $N = pq$  with primes  $p, q$  and integer  $r \geq 1$  [10, 16], we present a scheme producing garbled circuits of size  $|\widehat{C}| = O(|C|(\ell + k_{\text{DCR}}))$ , and input label of size  $|\mathbf{L}^x| = O(n(\ell + k_{\text{DCR}}))$ , where  $k_{\text{DCR}} = \log N$  is the bit-length of the modulus  $N$ . As such, the rate is just a constant  $O(1)$  when the integer values are sufficiently large, namely  $\ell = \Omega(k_{\text{DCR}})$ . To the best of our knowledge, this is the first garbling scheme for general unbounded depth circuits (in any model of computation) that achieve a constant rate, without relying on the strong primitive of iO (see Sect. 1.2 for a more detailed comparison).

**Theorem 1** (Informal, Arithmetic Garbling for Bounded Integer Computation). *Assume the DCR assumption over  $\mathbb{Z}_{N^{r+1}}^*$  for  $N = pq$  with primes  $p, q$  and  $r$  a sufficiently large positive integer. Let  $k_{\text{DCR}} = \lceil \log N \rceil$ ,  $B \in \mathbb{N}$ , and  $\ell = \lceil (\log 2B + 1) \rceil$ . There is an arithmetic garbling scheme for  $B$ -bounded integer computation, where the size of the garbled circuit is  $|\widehat{C}| = O(|C|(\ell + k_{\text{DCR}}))$  (i.e., rate  $O(1 + \frac{k_{\text{DCR}}}{\ell})$ ), and the length of input label is  $O(n(\ell + k_{\text{DCR}}))$  bits.*

**Part 2: Arithmetic Garbled Circuit over  $\mathbb{Z}_p$ .** Beyond bounded integer computations, can we support other important models of arithmetic computation? We consider *modular arithmetic computation* over a finite ring  $\mathcal{R} = \mathbb{Z}_p$  (where  $p$  is not necessarily a prime), which arises naturally in applications, in particular, in cryptosystems.

It turns out that the AIK CRT-based garbling scheme can be adapted to support  $\mathbb{Z}_p$ -computation<sup>3</sup>. However, as mentioned above, this solution does not

<sup>3</sup> This scheme reduces to Yao’s garbling by first decomposing the input elements into a bit representation using CRT. As such, this approach works as long as the inputs

satisfy many desiderata of arithmetic garbling, in particular, it makes non-black-box use of the Boolean circuit implementation of arithmetic operations. Though integer multiplication and mod- $p$  reduction are basic operations, there are actually many different algorithms (such as, Karasuba, Tom-Cook, Schönhage-Strassen, Barrett Reduction, Montgomery reduction to name a few), software implementation, and even hardware implementation. It is preferable to avoid applying cryptography to these algorithms/implementation, and have a modular design that can reap the benefits of any software/hardware optimization.

We present an arithmetic garbling scheme for  $\mathbb{Z}_p$ -computations, which makes only black-box call to the implementation of arithmetic operations.

**Theorem 2** (Informal, Arithmetic Garbling Scheme for Modular Computation). *Let  $p \in \mathbb{N}$  and  $\ell = \lceil \log p \rceil$ . There are arithmetic garbling schemes for computation over  $\mathbb{Z}_p$  that make only black-box use of implementation of arithmetic operations over  $\mathbb{Z}_p$ , as described below.*

- Assume DCR. The size of the garbled circuit is  $O(|C|(\ell + k_{\text{DCR}})k_{\text{DCR}})$  and the length of input labels is  $O(n\ell k_{\text{DCR}})$  bits (i.e., rate  $O(k_{\text{DCR}} + \frac{k_{\text{DCR}}^2}{\ell})$ ).
- Assume LWE with dimension  $k_{\text{LWE}}$ , modulus  $q$ , and noise distribution  $\chi$  that is poly( $k_{\text{LWE}}$ )-bounded, such that  $\log q = O(\ell) + \omega(\log k_{\text{LWE}})$ . The size of the garbled circuit is  $|C| \cdot \ell \cdot \tilde{O}(k_{\text{LWE}})$  and the length of input labels is  $\tilde{O}(n\ell k_{\text{LWE}})$  bits (i.e., rate  $\tilde{O}(k_{\text{LWE}})$ ).

We note that being black-box in the implementation of arithmetic operations, is different from being black-box in the ring. The latter has stringent conditions so that a construction that is black-box in the ring can automatically be applied to any ring. Unfortunately, Applebaum, Avron, and Brzuska [3] showed that such garbling is impossible for general circuits. Nevertheless, being black-box in the implementation of arithmetic operations already provides some of the benefits of a modular design. The garbler does not need to choose which algorithm/implementation of arithmetic operations to use, and evaluation can work with any algorithm/implementation.

**Part 3: Mixing Bounded Integer and Boolean Computation.** Many natural computational tasks mix arithmetic and Boolean computation. For example, a simple neural network component is a (fixed-point) linear functions fed into a ReLU activation functions, where  $\text{ReLU}(z) = \max(0, z)$  is much more efficient using (partially) Boolean computation. Even natural arithmetic computational tasks can benefit from (partial) boolean computation. Take the example of fast exponentiation: given  $(x, y)$  one can efficiently compute  $x^y$  if one has access to the bits of  $y$ ,  $y_\ell, \dots, y_0$  using the fact that  $x^y = x^{\sum_{i=0}^{\ell} y_i 2^i} = \prod_{i: y_i=1} x^{2^i}$ .

This motivates us to consider the following mixed model of computation, represented by a circuit consisting of three types of gates: 1) arithmetic operation gates  $+/-/\times : \mathcal{R}^2 \rightarrow \mathcal{R}$ , 2) Boolean function gates,  $g : \{0, 1\}^r \rightarrow \{0, 1\}^{r'}$ , where  $g$  is implemented using a Boolean circuit, and 3) the bit decomposition gate,

---

are integers from a bounded range and the computation can be implemented using Boolean circuits.

bits :  $\mathcal{R} \rightarrow \{0, 1\}^\ell$ , that maps a ring element to its bit representation. Naturally, a Boolean function gate can only take input from the bit decomposition gate or other Boolean function gates (otherwise, there is no restriction on how gates are connected).

We gave a construction for mixed bounded integer and Boolean computation. Our scheme naturally uses Yao’s garbled circuit to garble the Boolean function gates, and arithmetic garbled circuit (from Theorem 1 or AIK) to garble the arithmetic operation gates over bounded integers. Finally, we design a new gadget for bit decomposition, based on either DCR or LWE.

THE BIT DECOMPOSITION GADGET is an arithmetic garbling scheme for functions of form  $\text{BD}_{\{\mathbf{c}_j, \mathbf{d}_j\}}$  that maps an integer  $x \in [-B, B]$  to  $\ell$  labels, where the  $j$ ’th label is  $\mathbf{c}_j \text{bits}(x)_j + \mathbf{d}_j$ . This means given the garbled circuit  $\widehat{\text{BD}}$  and input label  $\mathbf{ax} + \mathbf{d}$ , the output labels are revealed and nothing else.

Our scheme puts together the above three components in a modular and black-box way. In terms of efficiency, the size of the garbled circuit naturally depends on the number of gates of each type. More specifically, garbling the Boolean computation gates incurs a rate of  $O(k_{\text{SKE}})$  inherited from Yao’s garbled circuit, whereas the arithmetic operation gates can be garbled with close to constant rate if using our DCR-based scheme in Theorem 1. Our bit decomposition gadget produces a garbled circuit of size  $O(\ell^2 \cdot k_{\text{DCR}})$  for sufficiently large integers  $\ell = \Omega(k_{\text{DCR}})$  if based on DCR, and of size  $\ell^2 \cdot \tilde{O}(k_{\text{LWE}})$  if based on LWE, where  $k_{\text{LWE}}$  is the LWE dimension. Recall that the AIK CRT-based scheme also relies on performing bit decomposition, however, at a much larger cost of  $O(\ell^6 k_{\text{SKE}})$ .

**Theorem 3** (Informal, Arithmetic Garbling Schemes for Mixed Computation). *Let  $B \in \mathbb{N}$  and  $\ell = \lceil \log 2B + 1 \rceil$ . There are arithmetic garbling schemes for mixed  $B$ -bounded integer and Boolean computation as described below.*

- Assume DCR. The size of the garbled circuit is  $O(s_b k_{\text{DCR}} + m_a(\ell + k_{\text{DCR}}) + m_b(\ell + k_{\text{DCR}})^2 \cdot k_{\text{DCR}})$ , where  $s_b$  is the total circuit size of all Boolean function gates,  $m_a$  the number of arithmetic operation gates, and  $m_b$  the number of bit-decomposition gates. The length of input label is  $O(n(\ell + k_{\text{DCR}}))$  bits.
- Assume LWE with dimension  $k_{\text{LWE}}$ , modulus  $q$ , and noise distribution  $\chi$  that is  $\text{poly}(k_{\text{LWE}})$ -bounded, such that  $\log q = O(\ell) + \omega(\log k_{\text{LWE}})$ . The size of the garbled circuit is  $s_b O(\lambda) + m_a \cdot \ell \cdot \tilde{O}(k_{\text{LWE}}) + m_b \cdot \ell^2 \cdot \tilde{O}(k_{\text{LWE}})$ . The length of input label is  $O(n\ell k_{\text{LWE}})$  bits, where  $\epsilon$  is a fixed constant (Fig. 2).

**Potential for Concrete Efficiency Improvement.** The primary goal of this work is designing new arithmetic garbling with good asymptotic efficiency. Though we do not focus on optimizing concrete efficiency, our DCR-based schemes do show potential towards practical garbling. Our concrete analysis demonstrates that when the input domains are large,  $\ell \sim k_{\text{DCR}} = 4096$  bits, the size of garbled circuits produced by our constant-rate bounded integer garbling scheme is significantly smaller than that of the Boolean baseline using the state-of-the-art Boolean garbling scheme of [17] – the garbling size of addition is  $\sim 100\times$  smaller, and the size of multiplication is  $\sim 500\times$  smaller. See Sect. 5.



**Table 2.** Summary of Our Garbling Schemes.

Computation	Assumption	Rate	Input Label Size
Bounded Arithmetic	DCR	$O(1 + k_{\text{DCR}}/\ell)$	$O(n(k_{\text{DCR}} + \ell))$
Mod $p$	DCR	$O(k_{\text{DCR}} + k_{\text{DCR}}^2/\ell)$	$O(nk_{\text{DCR}}\ell)$
Mod $p$	LWE	$\tilde{O}(k_{\text{LWE}})$	$\tilde{O}(n\ell k_{\text{LWE}})$
Mixed	DCR	$O((\ell + k_{\text{DCR}})k_{\text{DCR}})^*$	$O(n(\ell + k_{\text{DCR}}))$
Mixed	LWE	$\tilde{O}(\ell k_{\text{LWE}})^*$	$O(n\ell k_{\text{LWE}})$

\*Rate of Mixed Computation Schemes depends on relative frequency of gate types. Numbers here conservatively assume all gates are the most expensive type.

## 1.2 Related Works

We briefly survey approaches to garbling Boolean circuits that achieve good rate.

AIK showed that their LWE-based scheme when applied to constant-degree polynomials represented as a sum of monomials has constant-rate. The work of [2] yields a garbling scheme with size  $O(|C|) + \text{poly}(\lambda)$  and input size  $O(n + m + \text{poly}(\lambda))$ , assuming subexponentially secure indistinguishability obfuscation and rerandomizable encryption.

The work of [9, 11] presents a  $O(|C| + \text{poly}(\lambda, d))$ -size garbling of Boolean circuits, with input labels of size  $O(nm \text{poly}(\lambda, d))$  where  $d$  is the circuit depth,  $n$  is the input length,  $m$  is the output length, and  $\lambda$  the security parameter. One significant advantage of their scheme is that the circuit description is given in the clear. We analyze the sizes of garbled circuits and input labels when using their scheme to garble a  $B$ -bounded integer computation  $(C, x)$  of depth  $d$ , in particular, spelling out the exponent in the poly term. For simplicity of notation, we set the input length  $n$ , output length  $m$ , wire-value bit length  $\log B = \ell$ , and the size of a FHE bit encryption all to  $O(k)$ .

$$\begin{aligned}
 [9, 11]: & \quad |\tilde{C}| + |\mathbf{L}^x| > |C| + \tilde{O}(k^3 d^6 + k^6 d^4) \\
 \text{Our DCR-based scheme:} & \quad |\tilde{C}| + |\mathbf{L}^x| = O(|C|k)
 \end{aligned}$$

In comparison, the garbling of [9, 11] has smaller size when  $k$  and  $d$  are sufficiently small comparing with  $|C|$ , achieving even sub-constant rate  $O(|C|/k)$ . However, our garbled circuits are smaller when  $k$  and  $d$  are larger, achieving a constant rate for all  $k$  and  $d$ . The term  $\tilde{O}(k^3 d^6 + k^6 d^4)$  associated with [9, 11] is prohibitive, even for small  $k, d$  such as 100, whereas the complexity of our scheme does not have such large exponents. Our scheme is also simpler than [9, 11], which combines ABE, FHE, and Yao's garbled circuit in an intricate way.

The works of [7, 8] generalized FreeXOR [14], a technique that allows one to garble XOR gates at zero cost, to general arithmetic setting. They present a scheme for bounded integer computation where addition is for free. They also present a gadget (similar to our bit decomposition gadget) that converts integers to a primorial-mixed-radix representation, which has similar advantage as a Boolean representation (e.g. cheap comparisons). Leveraging free addition, they

show that their scheme has concrete performance benefit for certain bounded arithmetic computations, in comparison to directly applying Boolean garbling to arithmetic circuits. However, their construction is not arithmetic; in particular, the input encoding requires a “bit representation” of the inputs.

Finally, the work of [6] describes a method for generically shortening the length of input labels to  $|\mathbf{L}^x| = n\ell + o(n\ell)$  – that is, rate-1 input labels. However, the transformation does not preserve decomposability, which is a property that each input element  $x_i$  is encoded separately  $\mathbf{L}^i(x_i)$ . Many applications of garbling rely on decomposability, e.g., in 2PC, the party holding  $x_i$  can use OT/OLE to obtain  $\mathbf{L}^i(x_i)$ . The encoding of our schemes, AIK, and Yao’s garbled circuits all satisfy decomposability, and our DCR-based bounded integer garbling has the shortest input encoding (see Table 3).

### 1.3 Technical Overview

We start with reviewing the modular design paradigm of AIK, which is the basis of our approach.

As an arithmetic analog of Yao’s Boolean garbled circuits, the AIK garbling shares a similar high-level structure. Like Yao’s scheme, AIK’s scheme associates each wire value,  $x_i$ , with a wire label,  $\mathbf{L}^i$ , (which hides/encrypts the wire value).<sup>4</sup> Also like Yao’s scheme, the Garbler generates “garbled tables” that enable an evaluator holding a wire label for each input wire to a gate in the circuit to derive the corresponding output wire label. However, unlike in Yao’s scheme, the tables do not directly correspond to encryptions of the output wire labels under all possible input label pairs.

Instead, AIK builds bounded arithmetic garbled circuits in two steps: (1) they construct an information-theoretically secure garbling scheme for low depth arithmetic circuits over a ring  $\mathcal{R}$  (via black-box use of  $\mathcal{R}$ ), (2) they then construct a key extension gadget for bounded arithmetic computation that allows them to efficiently circumvent the depth restriction (the key extension gadget makes non-black-box use of  $\mathcal{R}$ , but the overall garbling scheme makes use of the key extension gadget in a black-box way).

To begin, let us recall how AIK construct (1) the information-theoretic scheme. This scheme does away with garbled gate information entirely, at the expense of long input labels whose structure depends explicitly on the circuit being garbled. In particular, for every wire of the circuit, the Garbler generates two keys  $\mathbf{k}_0^i, \mathbf{k}_1^i$  which are vectors in  $\mathcal{R}$ . During evaluation, for every wire, the evaluator should obtain a label  $\mathbf{L}^i = \mathbf{k}_0^i x_i + \mathbf{k}_1^i$  corresponding to the correct value of the wire as follows:

- *Input Labels*: For each input wire, its label is given to the evaluator.
- *Garbled Gate*: For every gate  $x_i = g(x_{j_1}, x_{j_2})$ , the invariant is that given the labels  $\mathbf{L}^{j_1}, \mathbf{L}^{j_2}$  corresponding to inputs  $x_{j_1}$  and  $x_{j_2}$ , the evaluator can

---

<sup>4</sup> In Yao’s scheme, these labels may be chosen independently and uniformly at random. In the arithmetic setting, this is infeasible as the domain may be exponentially large.

### Arithmetic Operation Gadgets

**Gadget for Addition**  $x_i = x_{j_1} + x_{j_2}$ : At garbling time, given a pair of keys  $(\mathbf{k}_0^i, \mathbf{k}_1^i)$  for the output wire  $i$ , it produces a pair of keys  $(\mathbf{k}_0^{j_1}, \mathbf{k}_1^{j_1})$  and  $(\mathbf{k}_0^{j_2}, \mathbf{k}_1^{j_2})$  for each input wire (and no garbled table) as follows:

$$\text{Set } \mathbf{k}_0^{j_1} = \mathbf{k}_0^{j_2} = \mathbf{k}_0^i \quad \text{Sample additive shares } \mathbf{k}_1^{j_1} + \mathbf{k}_1^{j_2} = \mathbf{k}_1^i .$$

At evaluation time, the output label can be obtained as follows

$$\mathbf{L}^{j_1} + \mathbf{L}^{j_2} = (\mathbf{k}_0^{j_1} x_{j_1} + \mathbf{k}_1^{j_1}) + (\mathbf{k}_0^{j_2} x_{j_2} + \mathbf{k}_1^{j_2}) = \mathbf{k}_0^i (x_{j_1} + x_{j_2}) + \mathbf{k}_1^i = \mathbf{L}^i .$$

**Gadget for Multiplication**  $x_i = x_{j_1} \times x_{j_2}$ : At garbling time, given output keys  $(\mathbf{k}_0^i, \mathbf{k}_1^i)$ , it produces input key pairs  $(\mathbf{k}_0^{j_1}, \mathbf{k}_1^{j_1})$  and  $(\mathbf{k}_0^{j_2}, \mathbf{k}_1^{j_2})$  (and no garbled table) as follows:

$$\mathbf{k}_0^{j_1} := (\mathbf{k}_0^i, s\mathbf{k}_0^i), \quad \mathbf{k}_1^{j_1} := (\mathbf{r}, \mathbf{u}), \quad \mathbf{k}_0^{j_2} := (\mathbf{1}, \mathbf{r}), \quad \mathbf{k}_1^{j_2} := (s, s\mathbf{r} - \mathbf{k}_1^i - \mathbf{u}) .$$

where  $s$  is a random scalar and  $\mathbf{r}, \mathbf{u}$  are random vectors.

At evaluation time, given input labels  $\mathbf{L}^{j_1} = (\mathbf{k}_0^i x_{j_1} + \mathbf{r}, s\mathbf{k}_0^i x_{j_1} + \mathbf{u})$  and  $\mathbf{L}^{j_2} = (x_{j_2} + s, \mathbf{r}x_{j_2} + s\mathbf{r} - \mathbf{k}_1^i - \mathbf{u})$ , the output label can be obtained as follows:

$$\mathbf{L}^i = \mathbf{L}_{\text{left}}^{j_1} \mathbf{L}_{\text{left}}^{j_2} - \mathbf{L}_{\text{right}}^{j_1} - \mathbf{L}_{\text{right}}^{j_2} .$$

**Fig. 1.** AIK Arithmetic Operation Gadgets

learn a label  $\mathbf{L}^i$  corresponding to the output  $x_i$  for each output wire, and no other information. This is achieved using the *arithmetic computation gadget* described in AIK, which are essentially information theoretically secure DARE (Decomposable Affine Randomized Encoding) for functions  $f_{+, \mathbf{k}_0^i, \mathbf{k}_1^i}(x_{j_1}, x_{j_2}) = \mathbf{k}_0^i(x_{j_1} + x_{j_2}) + \mathbf{k}_1^i$  and  $f_{\times, \mathbf{k}_0^i, \mathbf{k}_1^i}(x_{j_1}, x_{j_2}) = \mathbf{k}_0^i(x_{j_1} \times x_{j_2}) + \mathbf{k}_1^i$ . They are summarized in Fig. 1.<sup>5</sup>

*Remark: Having separate gadgets for addition and multiplication leaks the type of gate. There also exists an universal garbling gadget for arithmetic operation, which hides the gate operation, so that only the topology of the circuit is revealed.*

- *Outputs:* For each output wire, the evaluator learns  $\mathbf{L}^i = \mathbf{k}_0^i x_i + \mathbf{k}_1^i$ , which reveals the output  $x_i$  by setting  $\mathbf{k}_0^i = 1$  and  $\mathbf{k}_1^i = 0$ .

The above paradigm gives an information-theoretic arithmetic garbling scheme, however, only for logarithmic depth circuits. Its major issue is that the key-length increases exponentially in the depth of the circuit, because 1) the key-length of the input wires of a multiplication gate is twice the key-length of

<sup>5</sup> Note that while the evaluator can efficiently evaluate the garbled circuit from the bottom-up (inputs to outputs), the garbler (as described here) proceeds from the top-down: generating labels for the output wires and then recursively generating increasingly complex keys for the wire layers below.

its output wire, and 2) the key-length of input wires of any gate grows linearly with the fan-out that gate. On the flip side, this scheme has constant overhead for constant depth circuits.

To go beyond low-depth circuits, AIK introduced a *key-extension gadget*—a DARE for functions  $f_{\text{KE},\mathbf{c},\mathbf{d}}(x) = \mathbf{c} \cdot x + \mathbf{d}$ . It ensures that given the input label  $\mathbf{a} \cdot x + \mathbf{b}$  and garbled table, the evaluator can obtain a new *longer* label  $\mathbf{c} \cdot x + \mathbf{d}$ , and no other information. Now to support arbitrary depth circuit, AIK uses the arithmetic operation gadgets to handle the computation gates, and whenever the key length  $|\mathbf{c}|, |\mathbf{d}|$  becomes too long, it uses the key-extension gadget to shrink the key length down  $|\mathbf{a}|, |\mathbf{b}| < |\mathbf{c}|, |\mathbf{d}|$ .

It may seem counter-intuitive that a key “extension” gadget would be used to “shrink” keys, so let us discuss how this works in slightly more detail. First, recall that the information-theoretic DARE gadgets described in Fig. 1 derive (possibly longer) labels for the inputs to a gate from the output labels corresponding to that gate. Next, we break each wire  $i$  into two sub wires: the part that comes *out* of the preceding gate,  $i^{\text{out}}$ , and the part that goes *into* the next gate,  $i^{\text{in}}$  (for higher fan-out there will other  $i^{\text{in}}$  wires). By breaking up all wires in this manner, we can garble gates in parallel (as opposed to from the top-down) by independently and locally (a) sampling the (short) labels  $\mathbf{L}^{i^{\text{out}}}$ , and (b) locally applying the gadgets from Fig. 1 to derive (long) input labels  $\mathbf{L}^{j_1^{\text{in}}}, \mathbf{L}^{j_2^{\text{in}}}$ . At this point each wire value is now associated with two labels: a short output label and long input label(s). The key extension gadget allows the evaluator to derive the long input portion(s) from the short input label portion (using some extra information: the gabled table).

Therefore, this paradigm reduces the problem of constructing constant-overhead arithmetic garbling for bounded integer computation (Theorem 1) and arithmetic garbling for modular computation (Theorem 2) to the problem of designing (efficient) key-extension gadgets for the respective model of computation.

**Abstract Key-Extension Gadget.** Instead of describing AIK’s gadget, we will instead introduce an abstract approach to constructing key-extension gadgets (that also captures AIK’s key-extension gadget). Instantiating this approach has encountered significant technical barriers (discussed at length below), but we believe the high level paradigm is nonetheless instructive.

Recall that to construct a key-extension gadget the garbler knows *short* keys  $(\mathbf{a}, \mathbf{b})$  corresponding to *short* wire labels of the form  $\mathbf{S}_x = \mathbf{a} \cdot x + \mathbf{b}$  as well as *long* keys  $(\mathbf{c}, \mathbf{d})$  corresponding to *long* wire labels of the form  $\mathbf{L}_x = \mathbf{c} \cdot x + \mathbf{d}$ . The garbler’s task is to output some succinct information,  $\mathbf{tb}$ , so that an evaluator holding a short wire label  $\mathbf{S}_x$  can derive the long wire label corresponding to the same value  $\mathbf{L}_x$  without learning anything about the other wire labels  $\mathbf{L}_y$  (for  $y \neq x$ ).

As a warm up, observe that the Yao’s approach can be adapted to give an efficient key extension gadget for small domains. In particular for the boolean case of  $x \in \{0, 1\}$ , the garbler can simply set  $\mathbf{tb}$  to consist of two (one-time symmetric key) encryptions:  $\text{Enc}_{\mathbf{b}}(\mathbf{d}) = \text{Enc}_{\mathbf{S}_0}(\mathbf{L}_0)$  and  $\text{Enc}_{\mathbf{a}+\mathbf{b}}(\mathbf{c} + \mathbf{d}) = \text{Enc}_{\mathbf{S}_1}(\mathbf{L}_1)$

(randomly permuted). Using  $\mathbf{tb}$  the evaluator can simply decrypt the relevant ciphertext (using the short label as a key) to derive the long label corresponding to the same value. Semantic security implies that the evaluator learns nothing about the other label.

Unfortunately, it is not clear how to extend Yao’s approach to large arithmetic domains (with succinct garbled tables). Instead, it seems we need a stronger arithmetic properties from the encryption scheme. In particular, assume we have an encryption scheme,  $(\text{Enc}, \text{Dec})$ , which is *linearly homomorphic in both the key and message space*: there are operations  $\boxplus, \boxtimes$  such that  $x \boxtimes \text{Enc}_{\mathbf{a}}(\mathbf{c}) \boxplus \text{Enc}_{\mathbf{b}}(\mathbf{d}) = \text{Enc}_{\mathbf{a}+\mathbf{b}}(cx + \mathbf{d})$ .

Given such an encryption scheme, consider the case that the wire value  $x$  is *public* (we will relax this assumption momentarily). Then note that given a garbled table,  $\mathbf{tb}$ , comprised of just two cipher texts  $\text{Enc}_{\mathbf{a}}(\mathbf{c})$  and  $\text{Enc}_{\mathbf{b}}(\mathbf{d})$  the evaluator can use  $x, \mathbf{S}_x$  to derive a long label  $\mathbf{L}_x$  by homomorphically evaluating  $x \boxtimes \text{Enc}_{\mathbf{a}}(\mathbf{c}) \boxplus \text{Enc}_{\mathbf{b}}(\mathbf{d}) = \text{Enc}_{\mathbf{a}+\mathbf{b}}(cx + \mathbf{d}) = \text{Enc}_{\mathbf{S}_x}(\mathbf{L}_x)$  and decrypting. We need to additionally show that the evaluator learns nothing about the other output labels. In more detail, observe that we can simulate the view of evaluator holding  $\mathbf{S}_x, \mathbf{L}_x, x$  which is comprised of 3 cipher texts: (1)  $\text{Enc}_{\mathbf{a}}(\mathbf{c})$ , (2)  $\text{Enc}_{\mathbf{b}}(\mathbf{d})$ , and (3)  $\text{Enc}_{\mathbf{S}_x}(\mathbf{L}_x)$ . First, note that given  $\mathbf{L}_x, x$ , one can derive ciphertext (2) from ciphertexts (1) and (3) (and  $x$ ) by simply homomorphically computing  $\text{Enc}_{\mathbf{S}_x}(\mathbf{L}_x) \boxminus \text{Enc}_{\mathbf{a}}(\mathbf{c}) \boxtimes x = \text{Enc}_{\mathbf{b}}(\mathbf{d})$ . Armed with this observation we can invoke semantic security and simply simulate (given  $\mathbf{S}_x, \mathbf{L}_x, x$ ) by encrypting (3) honestly, replacing (1) with a random encryption, and homomorphically evaluating (2) from the other two ciphertexts.

There are two issues with this approach: the first (which we have already mentioned) is that the wire label is public, the second (and more subtle issue) is that we are implicitly assuming that encryption scheme has a key space that is identical to the message space which is in fact the ring  $\mathcal{R}$  we wish to compute over. We will describe a generic approach to dealing with the first issue here, but leave the second issue to the specific settings and implementations below.

We observe that one can effectively assume the wire label is public without loss of generality. The idea is that instead of extending the wire value  $x$  directly, we will mask  $x$  with a random value,  $r$ , that is known to the garbler to get  $x' = x + r$ . Note that  $x'$  can be safely output by the garbled circuit while statistically hiding  $x$ . Then we can use our key extension gadget to extend  $x'$ . Then once we have a long label  $\mathbf{L}_{x'}$  we can easily use another gadget to remove  $r$  (known to the garbler).<sup>6</sup>

**Key Extension Gadget for Bounded Integer Computation.** Our first key extension gadget relies on the Paillier extension of the Paillier encryption [10, 16]. This gadget is very efficient: the input label only consists of  $O(1)$  ring elements and the table size is proportional to the output label size.

We use a *one-time secure* version of the Paillier encryption. To generate the public parameters, sample two large safe primes and let  $N$  be the product of the

<sup>6</sup> Similar ideas are found in the well-known “half-gates” construction [19] of Zahur, Rosulek, and Evans for garbling boolean circuits comprised of XOR and AND gates.

two safe primes. Choose a small integer  $\zeta \geq 1$ , and the ciphertexts are vectors modulo  $N^{\zeta+1}$ . The group  $\mathbb{Z}_{N^{\zeta+1}}^*$  contains a hard subgroup of unknown order (i.e., the  $2N^\zeta$ 'th residue subgroup, the order of which is hard to compute given  $N$ ) and an easy subgroup of order  $N^\zeta$  generated by  $1 + N$ , in which discrete logarithm is easy. The public parameters are  $(N, \zeta, \mathbf{g})$ , where  $\mathbf{g} = (g_1, g_2, \dots, g_\psi)$  are randomly-sampled generators of the “hard” subgroup. The one-time use key  $s$  is an integer sampled uniformly from  $\{0, \dots, N\}$ . The encryption algorithm takes a message vector  $\mathbf{m} \in \mathbb{Z}_{N^\zeta}^\psi$  of dimension at most  $\psi$  as the input message, and generates a ciphertext as follows:

$$\text{Enc}(s, \mathbf{m}) = \mathbf{g}^s \cdot (1 + N)^{\mathbf{m}} = (g_1^s \cdot (1 + N)^{m_1}, \dots, g_\psi^s \cdot (1 + N)^{m_\psi}).$$

The Decisional Composite Residuosity (DCR) assumption implies that the ciphertext is pseudorandom. Indeed, the secret key can only be used once; in fact, the encryption algorithm is deterministic.

For our application, the following properties of the Paillier encryption are important:

- *Small Keys*: the secret key  $s$  is an integer upper bounded by  $N$  which is much smaller than the message space modulus  $N^\zeta$ .
- *Linear Homomorphism*: for any keys  $s_1, s_2 \in \mathbb{Z}$  and messages  $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}_{N^\zeta}^\psi$ ,

$$\text{Enc}(s_1, \mathbf{m}_1) \cdot \text{Enc}(s_2, \mathbf{m}_2) = \text{Enc}\left(\underbrace{s_1 + s_2}_{\text{over } \mathbb{Z}}, \underbrace{\mathbf{m}_1 + \mathbf{m}_2}_{\text{over } \mathbb{Z}_{N^\zeta}}\right).$$

In particular, given ciphertexts  $\text{Enc}(s_1, \mathbf{c}), \text{Enc}(s_2, \mathbf{d})$  and  $x$ , one can homomorphically compute  $\text{Enc}(s_1x + s_2, \mathbf{c}x + \mathbf{d})$ .

- *Integer Keys*: To decrypt the output ciphertext produced by the homomorphic evaluation, we need the key  $s_1x + s_2$ . Importantly, since the order of the hard group is unknown, we can only hope to use the key  $s_1x + s_2$  computed over  $\mathbb{Z}$ .

The above observations immediately suggest a naïve construction of key extension gadget: Let  $\text{Enc}(s_1, \mathbf{c}), \text{Enc}(s_2, \mathbf{d})$  be the garbled table, and  $(x, s_1x + s_2)$  computed over  $\mathbb{Z}$  be the input label. Decryption gives  $\mathbf{c} \cdot x + \mathbf{d} \bmod N^\zeta$  as desired. However, such a naïve construction faces two problems:

- *Input label over  $\mathbb{Z}$* . The output label is in ring  $\mathbb{Z}_{N^\zeta}$ . We will set  $N^\zeta \gg B$  to be sufficiently large so that a  $B$ -bounded computation can be “embedded” in computation modulo  $N^\zeta$ . As such, arithmetic operations can be garbled using AIK arithmetic operation gadgets in Fig. 1 with modulus  $N^\zeta$ . However, a problem is that to decrypt Paillier encryption, the input label  $s_1x + s_2$  must be computed over  $\mathbb{Z}$ . To close the gap, we crucially rely on the fact that in bounded integer computation, every wire value  $x$  is bounded. We can also sample  $s_1, s_2$  from a bounded range so that  $s_1x + s_2 < N^\zeta$ . Therefore, the input label can be  $(x, s_1x + s_2) \bmod N^\zeta = (x, s_1x + s_2)$  over  $\mathbb{Z}$ .
- *Leakage*. In the naïve construction,  $x$  is revealed. To hide  $x$ , we replace  $x$  by  $y = x + r$ , a one-time pad of  $x$ . Let  $(y, s_1y + s_2)$  be the input label, let  $\text{Enc}(s_1, \mathbf{c}), \text{Enc}(s_2, \mathbf{d} - \mathbf{r}\mathbf{c})$  be the table. The evaluator homomorphically computes  $\text{Enc}(s_1y + s_2, \mathbf{c}y + \mathbf{d} - \mathbf{r}\mathbf{c})$ , then decrypts  $\mathbf{c}y + \mathbf{d} - \mathbf{r}\mathbf{c} = \mathbf{c}x + \mathbf{d}$ .

For clarity, we sketch how this works. Say the wire value  $x$  is guaranteed to be bounded by  $-B \leq x \leq B$ . Sample  $r \leftarrow \{-B', \dots, B'\}$  for some  $B' \gg B$ , thus  $r + x$  statistically hides  $x$ . Sample  $s_1 \leftarrow \{0, \dots, N\}$ . Sample  $s_2 \leftarrow \{0, \dots, B''\}$  for some  $B'' \gg NB'$ , so that  $s_1(r + x) + s_2$  statistically hides  $s_1(r + x)$ , which in turn preserves semantic security for encryptions under  $s_1$ .<sup>7</sup> Choose  $\zeta$  so that  $N^\zeta > 2B''$ . Overall, the gadget consists of the following:

$$\begin{aligned} \text{Input Key: } \mathbf{a} &= (1, s_1) & \mathbf{b} &= (r, s_1 r + s_2) \\ \text{Input Label: } \mathbf{L}^{\text{in}} &= (r + x, s_1(r + x) + s_2) \\ \text{Garbled Table: } \text{Enc}(s_1, \mathbf{c}) & \quad \text{Enc}(s_2, \mathbf{d} - r\mathbf{c}) . \end{aligned}$$

We observe that the garbled table has “constant-rate”, which is the key leading to constant-rate garbled circuit. More precisely, the size of the above garbled table is  $|\mathbf{c}|(\zeta + 1) \log N$ . When the integer bound  $B$  is sufficiently large, it suffices to set the modulus  $N$  to be a constant times longer than  $B$ , i.e.,  $\log N = O(\log B)$ . In addition, the dimension of the output key  $|\mathbf{c}|$  is proportional to the fan out  $k$  of the wire with value  $x$ . Therefore, the garbled table has size  $|\mathbf{c}|(\zeta + 1) \log N = O(k \log B)$ , incurring a constant overhead. See Sect. 4 for more details.

**Key Extension Gadget for Modulo- $p$  Computation.** There are two barriers when we try to extend the previous key extension gadget to the modulo- $p$  computation setting.

- *Arbitrary Message Ring  $\mathbb{Z}_p$ .* In the Paillier encryption, the message is a vector over ring  $\mathbb{Z}_{N^\zeta}$ . It supports linear homomorphic evaluation modulo  $N^\zeta$ , where  $N$  is the product of two randomly sampled primes. But we need to perform computation modulo  $p$ , where  $p$  is an arbitrary integer specified by the given arithmetic circuit.
- *The Input Label over  $\mathbb{Z}$ .* The AIK arithmetic operations gadgets now uses keys and labels over  $\mathbb{Z}_p$ . However, as discussed above, to decrypt Paillier encryption, we need the input label  $s_1 y + s_2$  to be computed over  $\mathbb{Z}$ , where  $y$  now equals to  $(r + x) \bmod p$ . In the previous setting, we get around this problem easily because the wire value  $x$  is bounded, and hence computing  $s_1 y + s_2$  modulo  $N^\zeta$  is the same as computing it over the integers. Now, the wire value  $x$  could be an arbitrary element in  $\mathbb{Z}_p$ , certainly  $s_1 y + s_2 \bmod p$  is very different from  $s_1 y + s_2$  over  $\mathbb{Z}$ . We need a new technique to recover the latter.

To overcome the first barrier, we construct another encryption scheme on top of Paillier encryption, such that the message space is over  $\mathbb{Z}_p$ . The new encryption scheme is defined as

$$\overline{\text{Enc}}(s, \mathbf{m}) = \text{Enc}(s, \lfloor \mathbf{m} \cdot \frac{N^\zeta}{p} \rfloor), \quad \overline{\text{Dec}}(s, \mathbf{c}) = \lfloor \text{Dec}(s, \mathbf{c}) \cdot \frac{p}{N^\zeta} \rfloor.$$

<sup>7</sup> We do not need protect  $s_2$  because the corresponding ciphertext can be simulated using the ciphertext encrypted under  $s_1$  and the output label  $\mathbf{c}x + \mathbf{d}$ .

The new scheme satisfies a weaker form of linear homomorphism. Notice that for any  $m_1, m_2 \in \mathbb{Z}_p$ ,

$$\lfloor m_1 \cdot \frac{N^\zeta}{p} \rfloor + \lfloor m_2 \cdot \frac{N^\zeta}{p} \rfloor = \lfloor (m_1 + m_2) \cdot \frac{N^\zeta}{p} \rfloor + e$$

for some  $e \in \{-1, 0, 1\}$ . Therefore, for any  $s_1, s_2 \in \mathbb{Z}$  and  $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}_p^\psi$ ,

$$\overline{\text{Enc}}(s_1, \mathbf{m}_1) \cdot \overline{\text{Enc}}(s_2, \mathbf{m}_2) = \overline{\text{Enc}}\left(\underbrace{s_1 + s_2}_{\text{over } \mathbb{Z}}, \underbrace{\mathbf{m}_1 + \mathbf{m}_2}_{\text{modulo } p}\right) \cdot (1 + N)^{\mathbf{e}}$$

for some  $\mathbf{e} \in \{-1, 0, 1\}^\psi$ , and it can be correctly decrypted to  $\mathbf{m}_1 + \mathbf{m}_2$  given key  $s_1 + s_2$ , by simply decrypting according to Paillier and rounding the result to the nearest multiple of  $N^\zeta/p$ . The homomorphic evaluation can be extended to any linear function  $f(x_1, \dots, x_\ell) = c_1 x_1 + \dots + c_\ell x_\ell$ . For any  $s_1, \dots, s_\ell \in \mathbb{Z}$  and  $\mathbf{m}_1, \dots, \mathbf{m}_\ell \in \mathbb{Z}_p^\psi$ ,

$$\text{Dec}\left(f(s_1, \dots, s_\ell), \prod_{i=1}^{\ell} \overline{\text{Enc}}(s_i, \mathbf{m}_i)^{c_i}\right) = f(\mathbf{m}_1, \dots, \mathbf{m}_\ell)$$

as long as  $|f|_1 = \sum_i |c_i| \ll \frac{N^\zeta}{p}$ . Otherwise, if the magnitude of the coefficients are large, then the accumulation of the rounding error may break correctness.

In the main body, we also present an alternative construction of linear homomorphic encryption scheme based on the LWE assumption.

Now, using such a linear homomorphic encryption scheme (whose message space is over  $\mathbb{Z}_p$ ), we construct our key extension gadget: Sample random  $r \in \mathbb{Z}_p$  and let  $y = x + r \bmod p$  be the one-time pad of  $x$ . Sample random  $\mathbf{s}_1 \in \{0, 1\}^\ell$ ,  $\mathbf{s}_2 \in \{0, \dots, \lfloor p/2 \rfloor\}^\ell$ . We set the input label as

$$\mathbf{L}^{\text{in}} = (y, \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2) \bmod p.$$

Also define

$$\begin{aligned} \mathbf{s}_{\text{res}} &= \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \bmod p, \\ \mathbf{s}'_{\text{res}} &= \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \pmod{\mathbb{Z}}. \end{aligned}$$

Then  $\mathbf{L}^{\text{in}} = (y, \mathbf{s}_{\text{res}})$  and  $\mathbf{s}_{\text{res}} = \mathbf{s}'_{\text{res}} \bmod p$ .

Our key observation is that, given  $\mathbf{L}^{\text{in}} = (y, \mathbf{s}_{\text{res}})$ , one can recover  $\mathbf{s}'_{\text{res}}$ .

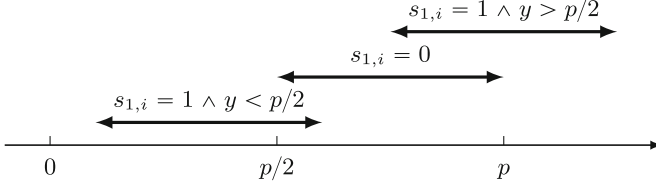
Let  $s_{\text{res},i}$  (resp.  $s'_{\text{res},i}, s_{1,i}, s_{2,i}$ ) denote the  $i$ -th coordinate of  $\mathbf{s}_{\text{res}}$  (resp.  $\mathbf{s}'_{\text{res}}, \mathbf{s}_1, \mathbf{s}_2$ ). Then

$$s'_{\text{res},i} = s_{1,i} y + (1 - s_{1,i}) \cdot \lfloor p/2 \rfloor + s_{2,i} = \begin{cases} y + s_{2,i}, & \text{if } s_{1,i} = 1, \\ \lfloor p/2 \rfloor + s_{2,i}, & \text{if } s_{1,i} = 0. \end{cases} \quad (1)$$

As illustrated by Fig. 2,

- In case  $y < p/2$ , we have  $0 \leq s'_{\text{res},i} < p$ , thus  $s'_{\text{res},i} = s_{\text{res},i}$ .





**Fig. 2.** The range of  $s'_{\text{res},i}$ , conditioning on  $s_{1,i}$  and  $y$

- In case  $y > p/2$ , we have  $\lfloor p/2 \rfloor \leq s'_{\text{res},i} < \lfloor p/2 \rfloor + p$ , thus  $s'_{\text{res},i}$  can also be recovered from  $s_{\text{res},i}$ .

Therefore,

$$s'_{\text{res},i} = \begin{cases} s_{\text{res},i} + p, & \text{if } y > p/2 \text{ and } s_{\text{res},i} < \lfloor p/2 \rfloor, \\ s_{\text{res},i}, & \text{otherwise.} \end{cases}$$

Since the evaluator can recover  $s'_{\text{res}} = \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2$ , if the table consists of

$$\text{“Enc}(\mathbf{s}_1, \mathbf{c})\text{” and “Enc}((1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2, \mathbf{d} - r\mathbf{c})\text{”}$$

then the evaluator can homomorphically compute  $\text{Enc}(s'_{\text{res}}, \mathbf{c}x + \mathbf{d})$  and decrypt it to get  $\mathbf{c}x + \mathbf{d}$ .

To formalize this idea, there are a few problems we have to overcome.

*Problem 1: Format Mismatch.* In the linear homomorphic encryption scheme, the key should be an integer sampled from a large interval. While  $\mathbf{s}_1$  is a vector consisting of 0's and 1's. To close the gap, we introduce a linear function  $\text{Lin} : \mathbb{Z}^\ell \rightarrow \mathbb{Z}$  to compress the length and to increase the magnitude. For example, if we let  $\text{Lin}(s_1, s_2, \dots, s_\ell) = s_1 + 2s_2 + 2^2s_3 + 2^3s_4 + \dots$ , then  $\text{Lin}(\mathbf{s}_1)$  is the uniform distribution over  $\{0, \dots, 2^\ell - 1\}$  since  $\mathbf{s}_1$  is sampled uniformly from  $\{0, 1\}^\ell$ .

Let the table be

$$\text{Enc}(\text{Lin}(\mathbf{s}_1), \mathbf{c}), \quad \text{Enc}(\text{Lin}((1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2), \mathbf{d} - r\mathbf{c}).$$

The evaluator homomorphically computes  $\text{Enc}(\text{Lin}(s'_{\text{res}}), \mathbf{c}x + \mathbf{d})$  and decrypts it to get  $\mathbf{c}x + \mathbf{d}$ .

After the introduction of  $\text{Lin}$ , the construction satisfies the correctness requirement. From now on, we will focus on the privacy issues.

*Problem 2: the Leakage of  $\mathbf{s}_1$ .* As shown by Eq. (1) and illustrated in Fig. 2,

$$\begin{aligned} s_{1,i} = 1 &\implies s'_{\text{res},i} \text{ is uniform in } [y, y + p/2), \\ s_{1,i} = 0 &\implies s'_{\text{res},i} \text{ is uniform in } [p/2, p). \end{aligned}$$

Therefore,  $s_{1,i}$  is hidden only if  $s'_{\text{res},i} \in [y, y + p/2) \cap [p/2, p)$ . Otherwise, when  $s'_{\text{res},i} \notin [y, y + p/2) \cap [p/2, p)$ , the value of  $s_{1,i}$  is leaked by  $s'_{\text{res},i}$ . For example, in the most extreme case when  $y = 0$ , the value of  $\mathbf{s}_1$  is completely leaked by  $s'_{\text{res}}$ .

We will later discuss how to repair the construction when  $y$  is close to zero. For now, let us assume  $y \in (p/4, 3p/4)$ . Under such assumption, for each  $i$ , there is a  $\geq 50\%$  chance that  $s_{1,i}$  is not revealed by  $\mathbf{s}'_{\text{res}}$ .

For privacy of the encryption scheme, we require that  $\text{Lin}(\mathbf{s}_1)$  is “sufficiently random” conditioning on  $\mathbf{s}'_{\text{res}}$ . In the full version we construct a (seeded) linear function  $\text{Lin}$ , such that with overwhelming probability,  $\text{Lin}(\mathbf{s}_1)$  *smudges*<sup>8</sup> the uniform distribution over  $\{0, \dots, N\}$ .

As analyzed in the full version let  $\text{Lin}(s_1, s_2, \dots, s_\ell) = \sum_i c_i s_i$ , where the coefficients  $c_1, \dots, c_\ell$  are i.i.d. sampled from  $\{0, \dots, N\}$ . Then as long as  $\ell \geq \log N$ ,  $\text{Lin}(\mathbf{s}_1)$  will smudge the uniform distribution over  $\{0, \dots, N\}$  even if about half of the coordinates of  $\mathbf{s}_1 \in \{0, 1\}^\ell$  are revealed. Here  $\text{Lin}$  is essentially a randomness extractor that is linear over  $\mathbb{Z}$ .

*Problem 3: the “Bad” Values of  $y$ .* So far, we have constructed a key extension gadget that works well when the one-time pad  $y = x + r \bmod p$  is in  $(p/4, 3p/4)$ , but it has serious privacy issue if  $y \in [0, p/4) \cup (3p/4, p)$ .

To close the leakage, we repeat the gadget one more time. This time use a different one-time pad  $\tilde{y} = x + r + \lfloor p/2 \rfloor \bmod p$ . Note that,  $y$  lies in the “bad” region  $[0, p/4) \cup (3p/4, p)$  if and only if  $\tilde{y}$  is in the “good” region  $(p/4, 3p/4)$ .

In greater detail, sample random  $r \in \mathbb{Z}_p$  and let

$$y = x + r \bmod p, \quad \tilde{y} = y + r + \lfloor p/2 \rfloor \bmod p.$$

Sample random  $\mathbf{s}_1, \tilde{\mathbf{s}}_1 \in \{0, 1\}^\ell$ ,  $\mathbf{s}_2, \tilde{\mathbf{s}}_2 \in \{0, \dots, \lfloor p/2 \rfloor\}^\ell$ , and let

$$\begin{aligned} \mathbf{s}_{\text{res}} &= \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \bmod p, \\ \tilde{\mathbf{s}}_{\text{res}} &= \tilde{\mathbf{s}}_1 \tilde{y} + (1 - \tilde{\mathbf{s}}_1) \cdot \lfloor p/2 \rfloor + \tilde{\mathbf{s}}_2 \bmod p, \\ \mathbf{s}'_{\text{res}} &= \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \text{ (over } \mathbb{Z}), \\ \tilde{\mathbf{s}}'_{\text{res}} &= \tilde{\mathbf{s}}_1 \tilde{y} + (1 - \tilde{\mathbf{s}}_1) \cdot \lfloor p/2 \rfloor + \tilde{\mathbf{s}}_2 \text{ (over } \mathbb{Z}). \end{aligned}$$

Set  $\mathbf{L}^{\text{in}} = (y, \mathbf{s}_{\text{res}}, \tilde{\mathbf{s}}_{\text{res}})$  as the input label. Let  $(\mathbf{c}_1, \mathbf{d}_1), (\mathbf{c}_2, \mathbf{d}_2)$  be additive sharings of  $(\mathbf{c}, \mathbf{d})$ . The table consists of

$$\begin{aligned} \text{Enc}(\text{Lin}(\mathbf{s}_1), \mathbf{c}_1), & \quad \text{Enc}(\text{Lin}((1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2), \mathbf{d}_1 - r\mathbf{c}_1), \\ \text{Enc}(\text{Lin}(\tilde{\mathbf{s}}_1), \mathbf{c}_2), & \quad \text{Enc}(\text{Lin}((1 - \tilde{\mathbf{s}}_1) \cdot \lfloor p/2 \rfloor + \tilde{\mathbf{s}}_2), \mathbf{d}_2 - r\mathbf{c}_2). \end{aligned}$$

Given the table and input label, the evaluator homomorphically evaluates  $\text{Enc}(\text{Lin}(\mathbf{s}'_{\text{res}}), \mathbf{c}_1 x + \mathbf{d}_1)$ ,  $\text{Enc}(\text{Lin}(\tilde{\mathbf{s}}'_{\text{res}}), \mathbf{c}_2 x + \mathbf{d}_2)$ . The evaluator recovers  $\mathbf{s}'_{\text{res}}, \tilde{\mathbf{s}}'_{\text{res}}$  from the input label, and decrypts both ciphertexts to get  $\mathbf{c}_1 x + \mathbf{d}_1, \mathbf{c}_2 x + \mathbf{d}_2$ . In the end, output  $\mathbf{L}^{\text{out}} = \mathbf{c}x + \mathbf{d} = (\mathbf{c}_1 x + \mathbf{d}_1) + (\mathbf{c}_2 x + \mathbf{d}_2) \bmod p$ .

**Bit Decomposition Gadget.** Besides purely arithmetic computation, we also consider a computation model that combines Boolean operations and arithmetic operation. Garbling such mixed computation is enabled by the *bit decomposition*

<sup>8</sup> Formally,  $\text{Lin}(\mathbf{s}_1)$  smudges the uniform distribution over  $\{0, \dots, N\}$  if  $\text{Lin}(\mathbf{s}_1)$  and  $\text{Lin}(\mathbf{s}_1) + u$  are statistically indistinguishable, where  $u$  is sampled from  $\{0, \dots, N\}$ .

*gadget* — a DARE for functions  $f_{\text{BD},\{\mathbf{c}_j,\mathbf{d}_j\}}(x) = \{\mathbf{c}_j \text{bits}(x)_j + \mathbf{d}_j\}$ . It ensures that given  $\mathbf{a}x + \mathbf{b}$  and the garbled table, the evaluator can get a label  $\mathbf{c}_j \text{bits}(x)_j + \mathbf{d}_j$  for every bit in the bit representation of  $x$ .

Notice that, in order to build the bit decomposition gadget, it suffices to design the *truncation gadget*. Let  $\lfloor x \rfloor_{2^j} := \lfloor x/2^j \rfloor$  denotes the integer quotient of  $x$  divided by  $2^j$ . This operation truncates  $j$  least significant bits. The truncation gadget is a DARE for functions  $f_{\text{TC},\mathbf{c},\mathbf{d}}(x) = \mathbf{c} \cdot \lfloor x \rfloor_2 + \mathbf{d}$ . Given a label of  $x$  and the garbled table, the evaluator can get a label for the truncated value  $\lfloor x \rfloor_2$ . Once we have the truncation gadget, the evaluator can use the truncation gadgets  $j$  times to get a label for the truncated value  $\lfloor x \rfloor_{2^j}$  for every  $j$ . Thus the evaluator can compute a label of the  $j$ -th bit of  $x$  via

$$\underbrace{\mathbf{c} \lfloor x \rfloor_{2^{j-1}} + \mathbf{d}_1}_{\text{a label of } \lfloor x \rfloor_{2^{j-1}}} - 2 \cdot \underbrace{(\mathbf{c} \lfloor x \rfloor_{2^j} + \mathbf{d}_2)}_{\text{a label of } \lfloor x \rfloor_{2^j}} = \underbrace{\mathbf{c} \cdot \text{bits}(x)_j + (\mathbf{d}_1 - 2\mathbf{d}_2)}_{\text{a label of the } j\text{-th bit of } x}.$$

Now the task has been reduced to designing the truncation gadget. Our construction of the truncation gadget is inspired by the techniques used in the key extension gadgets. The first idea is to sample random  $r$  from a sufficiently large range, and to consider the one-time pad  $y = x + r$ . Instead of generating the labels of  $\text{bits}(x)_j$ , we construct an (imperfect) bit decomposition gadget that generates the labels of each  $\text{bits}(y)_j$ . Once evaluator has the labels of every bit of  $y$ , it can compute the labels of every bit of  $x$ , as long as we additionally give the evaluator a Yao's Boolean garbled circuit, with  $r$  hard-coded inside. Thus correspondingly, it suffices to construct an (imperfect) truncation gadget that allows the evaluator to get  $\mathbf{c} \lfloor y \rfloor_2 + \mathbf{d}$ .

Inspired by our key extension gadget for modulo- $p$  computation, the gadget table of the (imperfect) truncation gadget looks like

$$\text{Enc}(\text{Lin}(\mathbf{s}_1), \mathbf{c}), \quad \text{Enc}(\text{Lin}(\lfloor \mathbf{s}_2 \rfloor_2), \mathbf{d}).$$

The evaluator can homomorphically evaluate  $\text{Enc}(\text{Lin}(\mathbf{s}_1 \lfloor y \rfloor_2 + \lfloor \mathbf{s}_2 \rfloor_2), \mathbf{c} \lfloor y \rfloor_2 + \mathbf{d})$ .

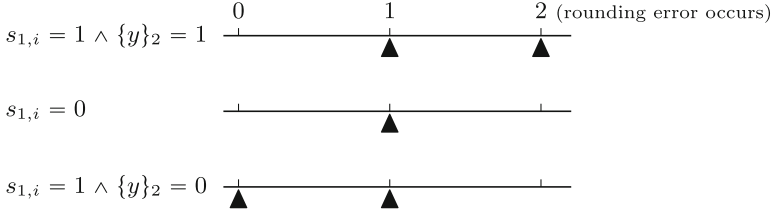
The input label of the truncation gadget is

$$(y, \mathbf{s}_1 y + \mathbf{s}_2), \quad \text{which equals } (x + r, \mathbf{s}_1 x + (\mathbf{s}_1 r + \mathbf{s}_2)).$$

If the evaluator can recover  $\mathbf{s}_1 \lfloor y \rfloor_2 + \lfloor \mathbf{s}_2 \rfloor_2$  from the input label, it can decrypts  $\mathbf{c} \lfloor y \rfloor_2 + \mathbf{d}$  using the key  $\text{Lin}(\mathbf{s}_1 \lfloor y \rfloor_2 + \lfloor \mathbf{s}_2 \rfloor_2)$ .

To enable the recovery,  $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{Z}^\ell$  are sampled from carefully chosen distributions.  $\mathbf{s}_1$  is sampled uniformly from  $\{0, 1\}^\ell$ .  $\mathbf{s}_2$  is sampled conditioning on  $\mathbf{s}_1$ : for each  $i \leq \ell$ ,

$$s_{2,i} = \begin{cases} \text{a random integer in } [0, B_{\text{smdg}}), & \text{if } s_{1,i} = 1 \\ \text{a random odd integer in } [0, B_{\text{smdg}}), & \text{if } s_{1,i} = 0 \end{cases}$$



**Fig. 3.** The range of  $\{s_{1,j}y\}_2 + \{s_{2,j}\}_2$ , conditioning on  $s_{1,i}$  and  $\{y\}_2$

where  $B_{\text{smdeg}}$  is a sufficiently large bound. We sample  $\mathbf{s}_1, \mathbf{s}_2$  in such a way to ensure that  $s_{1,j} \lfloor y \rfloor_2 + \lfloor s_{2,j} \rfloor_2$  can be recovered from  $(y, s_{1,j}y + s_{2,j})$ .

Given  $(y, s_{1,j}y + s_{2,j})$ , the evaluator can compute  $\lfloor s_{1,j}y + s_{2,j} \rfloor_2$ , which is very close to the target value. In particular,

$$s_{1,j} \lfloor y \rfloor_2 + \lfloor s_{2,j} \rfloor_2 = \begin{cases} \lfloor s_{1,j}y + s_{2,j} \rfloor_2 - 1, & \text{if both } s_{1,j}y \text{ and } s_{2,j} \text{ are odd} \\ \lfloor s_{1,j}y + s_{2,j} \rfloor_2, & \text{otherwise} \end{cases}$$

The evaluator can offset the error if it can tell whether both  $s_{1,j}y$  and  $s_{2,j}$  are odd. We claim:

$$\text{both } s_{1,j}y \text{ and } s_{2,j} \text{ are odd} \iff y \text{ is odd and } s_{1,j}y + s_{2,j} \text{ is even,} \quad (2)$$

By this, the evaluator can recover  $s_1 \lfloor y \rfloor_2 + \lfloor s_2 \rfloor_2$ .

The claim (2) can be proved by enumerating the possible parities of  $s_{1,j}, y, s_{2,j}$ . We also provide a visualized proof of this claim. Let  $\{z\}_2 := z - 2 \lfloor z \rfloor_2$  denote the remainder of  $z$  divided by 2. The rounding error occurs if and only if  $\{s_{1,j}y\}_2 + \{s_{2,j}\}_2 = 2$ . As shown by Fig. 3, when  $y$  is even, there is no rounding error; when  $y$  is odd, the rounding error occurs only if  $s_{1,j}y + s_{2,j}$  is even.

Figure 3 also shows that  $s_{1,j}$  is not always hidden by  $s_{1,j}y + s_{2,j}$ . For privacy, we require that  $\text{Lin}(\mathbf{s}_1)$  is “sufficiently random” even conditioning on the leakage. Such (seeded) linear function  $\text{Lin}$  is constructed in the full version.

**Organization.** In Sect. 2, we define three models of computations, bounded integer, modular arithmetic, and mixed computation, our garbling scheme, and the key extension gadget. The full version also defines the arithmetic computation and bit decomposition gadgets. In Sect. 3, we introduce a linearly homomorphic encryption scheme (LHE) as a tool for constructing the gadgets. In the full version, we instantiate it under either the DCR or the LWE assumption. In Sect. 4, we construct a key extension gadget in the bounded integer model. The full version also constructs key extension in the modular arithmetic model, bit decomposition in the mixed model, and the overall garbling scheme in all three models. In Sect. 5, we compare the concrete efficiency of our scheme, in the bounded integer model, with the scheme of [8] and the Boolean baseline using [17]. The full version extends the comparison to the modular arithmetic and the mixed models.

## 2 Definitions

A circuit over some domain  $\mathcal{I} \subseteq \mathbb{Z}$  consists of connected gates that each computes some function over  $\mathcal{I}$ . For a circuit  $C$  with  $n$  input wires and a vector  $\mathbf{x} \in \mathcal{I}^n$ ,  $(C, \mathbf{x})$  is referred to as a computation.

In the following, we define three classes of circuits by specifying their respective domains, allowed types of gates, and admissible inputs. Each class of circuits is also referred to as a model of computation.

**Modular Arithmetic Computation.** In this model, a circuit  $C$  consists of three types of gates: addition, subtraction, and multiplication over  $\mathbb{Z}_p$  (all with fan-in two). Its domain is simply  $\mathcal{I} = \mathbb{Z}_p$ . That is, every input and intermediate computation value is in  $\mathbb{Z}_p$ . For a circuit  $C$  with  $n$  inputs, all input vectors in  $\mathbb{Z}_p^n$  are admissible.

**Bounded Integer Computation.** In this model, a circuit  $C$  consists of the same arithmetic gates as above, computed over  $\mathbb{Z}$ . Its domain is the set of integers whose absolute values are bounded by some positive integer  $B$ , denoted as  $\mathcal{I} = \mathbb{Z}_{\leq B}$ . For a circuit  $C$ , an input vector is admissible if and only if  $(C, \mathbf{x})$  is  $B$ -bounded, i.e., every input and intermediate computation value while evaluating  $C(\mathbf{x})$  is in the range  $[-B, B]$ .

**Mixed Bounded Integer and Boolean Computation.** This model extends bounded integer computation, with domain  $\mathcal{I} = \mathbb{Z}_{\leq B}$  and bit length  $d = \lceil \log(2B + 1) \rceil$ , to include the following additional gates.

- The bit decomposition gate  $g_{BD} : \mathbb{Z}_{\leq B} \rightarrow \{0, 1\}^d$  is defined by  $g_{BD}(x) = \text{bits}(x)_1, \dots, \text{bits}(x)_d$ , where  $\text{bits}(x)_i$  represents the  $i^{\text{th}}$  bit  $x$ . By default, we let  $\text{bits}(x)_d$  represent the “sign” of  $x$ : for a non-negative integer  $x$ ,  $\text{bits}(x)_d = 0$ , and for a negative integer  $x$ ,  $\text{bits}(x)_d = 1$ . The rest of the bits represent the magnitude of  $x$  such that  $|x| = \sum_{i=1}^{d-1} 2^{i-1} \text{bits}(x)_i$ .

The output of  $g_{BD}$  can be used in two ways. First, they can be interpreted as 0, 1 values in  $\mathbb{Z}_{\leq B}$ , and fed into further arithmetic computations. Second, they can be used as inputs to other Boolean computation gates  $g : \{0, 1\}^{d_1} \rightarrow \{0, 1\}^{d_2}$ . In general, we allow any Boolean computation gate  $g$  that can be computed by a polynomial-size Boolean circuit. Interesting examples include comparison and truncation.

- A comparison gate  $g_{\text{comp}} : \{0, 1\}^d \times \{0, 1\}^d \rightarrow \{0, 1\}$  is defined as  $g_{\text{comp}}(\text{bits}(x), \text{bits}(y)) = 1$  iff  $x > y$ .
- A truncation gate  $g_{\text{trun}}^\Delta : \{0, 1\}^d \rightarrow \{0, 1\}^d$  with parameter  $\Delta \in \mathbb{Z}_{\leq B}$  is defined as  $g_{\text{comp}}(\text{bits}(x)) = \text{bits}(\lfloor \frac{x}{\Delta} \rfloor)$ .

Formally, we define classes of polynomial-sized circuits in above-described models: Let  $\mathcal{C}_{\mathbb{Z}_p}^{\text{Arith}} = \{\mathcal{C}_{\mathbb{Z}_{p(\lambda)}, \lambda}^{\text{Arith}}\}_\lambda$ ,  $\mathcal{C}_{\mathbb{Z}_{\leq B}}^{\text{BI}} = \{\mathcal{C}_{\mathbb{Z}_{\leq B(\lambda)}, \lambda}^{\text{BI}}\}_\lambda$ , and  $\mathcal{C}_{\mathbb{Z}_{\leq B}}^{\text{BI-decomp}} = \{\mathcal{C}_{\mathbb{Z}_{\leq B(\lambda)}, \lambda}^{\text{BI-decomp}}\}_\lambda$  contain circuits consisting of a polynomial number of gates in respectively the modular arithmetic computation with modulus  $p(\lambda)$ ,  $B(\lambda)$ -bounded integer computation, and  $B(\lambda)$ -bounded integer and Boolean computation model. The bound  $B(\lambda)$  and modulus  $p(\lambda)$  are bounded by  $2^{\text{poly}(\lambda)}$  for

some fixed polynomial. When talking about a general model of computation, we will use the notation  $\mathcal{C} \in \{\mathcal{C}_{\mathbb{Z}_p}^{\text{Arith}}, \mathcal{C}_{\mathbb{Z}_{\leq B}}^{\text{BI}}, \mathcal{C}_{\mathbb{Z}_{\leq B}}^{\text{BI-decomp}}\}$  over  $\mathcal{I} \in \{\mathbb{Z}_p, \mathbb{Z}_{\leq B}\}$ .

**Notations for Garbling.** For a model of computation  $\mathcal{C}$  over  $\mathcal{I}$ , our garbling scheme introduces two more spaces: a label space  $\mathcal{L}$ , and a ciphertext space  $\mathcal{E}$ , where  $\mathcal{I} \subseteq \mathcal{L}$ .

Similar to prior garbling schemes [5], the garbling algorithm assigns two keys  $\mathbf{z}_1, \mathbf{z}_2 \in \mathcal{L}^\ell$  of dimension  $\ell$  to each wire in a computation  $(C, \mathbf{x})$ . If this wire has a value  $x \in \mathcal{I}$ , then the evaluator should obtain a label  $\mathbf{L} = \mathbf{z}_1 x + \mathbf{z}_2$  computed over  $\mathcal{L}$  (by interpreting  $x \in \mathcal{I}$  as elements in  $\mathcal{L}$ ).

For each gate in  $C$ , the garbling algorithm outputs a garbled table consisting of some ciphertexts in  $\mathcal{E}$ . These ciphertexts, together with labels for the input wires, allow an evaluator to obtain a label for each of its output wires.

## 2.1 Definition of Garbling Schemes

**Definition 1** (garbling). *Let  $\mathcal{C} \in \{\mathcal{C}_{\mathbb{Z}_p}^{\text{Arith}}, \mathcal{C}_{\mathbb{Z}_{\leq B}}^{\text{BI}}, \mathcal{C}_{\mathbb{Z}_{\leq B}}^{\text{BI-decomp}}\}$  be a model of computation over the domain  $\mathcal{I} \in \{\mathbb{Z}_p, \mathbb{Z}_{\leq B}\}$ . A garbling scheme for  $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$  over  $\mathcal{I} = \mathcal{I}(\lambda)$ , with a label space  $\mathcal{L} = \mathcal{L}(\lambda)$  consists of three efficient algorithms.*

- $\text{Setup}(1^\lambda)$  takes a security parameter  $\lambda$  as input, and outputs public parameters  $\text{pp}$ , which define a ciphertext space  $\mathcal{E}$ , and specify a polynomial dimension  $\ell$  for keys and labels.  
The rest of the algorithms have access to  $\text{pp}$ .
- $\text{Garble}^{\text{pp}}(1^\lambda, 1^\ell, C)$  takes as inputs a security parameter  $\lambda$ , and a circuit  $C \in \mathcal{C}_\lambda$  with input length  $n$ . It outputs  $n$  key pairs  $\{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i \in [n]} \in \mathcal{L}^\ell$  of dimension  $\ell$  specified by  $\text{pp}$ , independent of the circuit size  $|C|$ , and a garbled circuit  $\widehat{C}$  (consisting of many garbled tables, each further contains ciphertexts in  $\mathcal{E}$ ).
- $\text{Dec}^{\text{pp}}(\{\mathbf{L}^i\}_{i \in [n]}, \widehat{C})$  takes as inputs  $n$  labels  $\mathbf{L}^i \in \mathcal{L}^\ell$ , and a garbled circuit  $\widehat{C}$ . It outputs an evaluation result  $y \in \mathcal{I}$ .

**Correctness.** *The scheme is correct if for all  $\lambda \in \mathbb{N}$ ,  $\text{pp} \in \text{Supp}(\text{Setup}(1^\lambda))$ , circuit  $C \in \mathcal{C}_\lambda$  with  $n$  input wires, and input  $\mathbf{x} = (x_1, \dots, x_n) \in \mathcal{I}^n$  that's admissible to  $C$ , the following holds.*

$$\Pr \left[ \begin{array}{l} \text{Dec}^{\text{pp}}(\{\mathbf{L}^i\}_{i \in [n]}, \widehat{C}) \\ = C(\mathbf{x}) \text{ (over } \mathcal{I}) \end{array} \middle| \begin{array}{l} \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i \in [n]}, \widehat{C} \leftarrow \text{Garble}^{\text{pp}}(1^\lambda, C), \\ \mathbf{L}^i = \mathbf{z}_1^i x_i + \mathbf{z}_2^i \text{ (over } \mathcal{L}) \end{array} \right] = 1.$$

**Security.** *A simulator  $\text{Sim}$  for the garbling scheme has following syntax.*

- $\text{Sim}(1^\lambda, \text{pp}, C, y)$  takes as inputs a security parameter  $\lambda$ , public parameters  $\text{pp}$ , a circuit  $C \in \mathcal{C}_\lambda$ , and an evaluation result  $y \in \mathcal{I}$ . It outputs  $n$  simulated labels  $\{\widetilde{\mathbf{L}}^i\}_{i \in [n]}$  and a simulated garbled circuit  $\widetilde{C}$ .

The garbling scheme is secure if there exists an efficient simulator  $\text{Sim}$  such that for all sequence of circuits  $\{C_\lambda\}_\lambda$  where each  $C_\lambda \in \mathcal{C}_\lambda$  has  $n = n(\lambda)$  inputs, and sequence of admissible inputs  $\{\mathbf{x}_\lambda\}_\lambda$  where  $\mathbf{x}_\lambda = (x_{1,\lambda}, \dots, x_{n,\lambda}) \in \mathcal{I}^n$ , the following indistinguishability holds. (We suppress the index  $\lambda$  below.)

$$\approx_c \left\{ \text{pp}, \{\mathbf{L}_i\}_{i \in [n]}, \widehat{C} \right\} \left| \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda), \\ \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i \in [n]}, \widehat{C} \leftarrow \text{Garble}^{\text{PP}}(1^\lambda, C), \\ \mathbf{L}_i = \mathbf{z}_1^i x_i + \mathbf{z}_2^i, y = C(\mathbf{x}) \end{array} \right.$$

Recall that in bounded integer computations (i.e.,  $\mathcal{C} = \mathcal{C}_{\mathbb{Z} \leq B}^{\text{BI}}$  or  $\mathcal{C}_{\mathbb{Z} \leq B}^{\text{BI-decomp}}$ ), an input  $\mathbf{x}$  is admissible to a circuit  $C$  if and only if  $(C, \mathbf{x})$  is  $B$ -bounded. In modular arithmetic computations (i.e.,  $\mathcal{C} = \mathcal{C}_{\mathbb{Z}_p}^{\text{Arith}}$ ) all inputs are admissible.

## 2.2 Definition of Garbling Gadgets

Our garbling scheme garbles a circuit in a gate-by-gate fashion. To handle different types of gates, we introduce different garbling gadgets. In addition to the arithmetic computation gates, bit decomposition gates, and general Boolean computation gates as introduced earlier, we also consider the following key extension gates, which are artificially added to every circuit at garbling time.

**Key Extension Gate** has one input and one output wire and implements the identity function  $f(x) = x$ . Inserting this gate anywhere in a circuit does not change the function computed. However, garbling the key extension gate has the following effect during evaluation: Given a short label for the input wire  $\mathbf{z}_1^{\text{in}} x + \mathbf{z}_2^{\text{in}}$  of dimension  $\ell$ , the evaluator can obtain a much longer label for the output wire  $\mathbf{z}_1^{\text{out}} x + \mathbf{z}_2^{\text{out}}$  of some dimension  $\ell' > \ell$ .

Our key extension gadget for handling the above gate is exactly the “key shrinking” gadget in [5], and is the technical core of this work. We define it formally below. The analogous definitions of the arithmetic computation, bit composition, and Boolean computation gadgets are deferred to the full version.

**Gadgets Share Setup of Garbling Scheme.** Each gadget is defined with respect to a garbling scheme for some model of computation  $\mathcal{C}$  over a domain  $\mathcal{I}$  and a label space  $\mathcal{L}$ . The gadget depends on the public parameters  $\text{pp}$  generated by the  $\text{Setup}$  algorithm of the garbling scheme, which specifies a ciphertext space  $\mathcal{E}$ , and a key dimension  $\ell \in \mathbb{N}$ . Its algorithms all have random access to  $\text{pp}$ .

**Key Extension Gadget.** The key extension gadget consists of three algorithms,  $\text{KeyGen}$ ,  $\text{Garble}$ , and  $\text{Dec}$ . To handle a key extension gate, we assume each of its output wires is already assigned an output key pair. Their concatenation form a long “target” key pair  $\mathbf{z}_1^{\text{out}}, \mathbf{z}_2^{\text{out}}$ . At garbling time, the garbler uses  $\text{KeyGen}$ ,  $\text{Garble}$  to generate a short key pair  $\mathbf{z}_1^{\text{in}}, \mathbf{z}_2^{\text{in}}$  for the input wire, and a garbled table  $\text{tb}$ . At evaluation time, the evaluator uses  $\text{Dec}$  on the input label  $\mathbf{L}^{\text{in}} = \mathbf{z}_1^{\text{in}} x + \mathbf{z}_2^{\text{in}}$  for some value  $x$ , and the garbled table  $\text{tb}$  to recover the output label  $\mathbf{L}^{\text{out}} = \mathbf{z}_1^{\text{out}} x + \mathbf{z}_2^{\text{out}}$ . We define the algorithms formally below.

**Definition 2** (key extension).

- $\text{KE.KeyGen}^{\text{PP}}(1^\lambda, 1^\ell)$  takes as inputs a security parameter  $\lambda$ , and the key dimension  $\ell$  specified by  $\text{pp}$ . It samples a key pair  $\mathbf{z}_1^{\text{in}}, \mathbf{z}_2^{\text{in}} \in \mathcal{L}^\ell$ .
- $\text{KE.Garble}^{\text{PP}}(\mathbf{z}_1^{\text{out}}, \mathbf{z}_2^{\text{out}}, \mathbf{z}_1^{\text{in}}, \mathbf{z}_2^{\text{in}})$  takes as inputs a (long) key pair  $\mathbf{z}_1^{\text{out}}, \mathbf{z}_2^{\text{out}} \in \mathcal{L}^{\ell'}$ , and a (short) key pair  $\mathbf{z}_1^{\text{in}}, \mathbf{z}_2^{\text{in}} \in \mathcal{L}^\ell$ . It outputs a garbled table  $\text{tb}$  (consisting of many ciphertexts in  $\mathcal{E}$ ).
- $\text{KE.Dec}^{\text{PP}}(\mathbf{L}^{\text{in}}, \text{tb})$  takes as inputs a short label  $\mathbf{L}^{\text{in}} \in \mathcal{L}^\ell$  and a garbled table  $\text{tb}$ . It outputs a long label  $\mathbf{L}^{\text{out}} \in \mathcal{L}^{\ell'}$ .

**Correctness.** The scheme is correct if for all  $\lambda \in \mathbb{N}$ ,  $\text{pp} \in \text{Supp}(\text{Setup}(1^\lambda))$ ,  $\mathbf{z}_1^{\text{out}}, \mathbf{z}_2^{\text{out}} \in \mathcal{L}^{\ell'}$  of dimension  $\ell' \in \mathbb{N}$ , and  $x \in \mathcal{I}$ , the following holds.

$$\Pr \left[ \begin{array}{l} \text{KE.Dec}^{\text{PP}}(\mathbf{L}^{\text{in}}, \text{tb}) \\ = \mathbf{L}^{\text{out}} \end{array} \middle| \begin{array}{l} \mathbf{z}_1^{\text{in}}, \mathbf{z}_2^{\text{in}} \leftarrow \text{KE.KeyGen}^{\text{PP}}(1^\lambda, 1^\ell), \\ \text{tb} \leftarrow \text{KE.Garble}^{\text{PP}}(\mathbf{z}_1^{\text{out}}, \mathbf{z}_2^{\text{out}}, \mathbf{z}_1^{\text{in}}, \mathbf{z}_2^{\text{in}}), \\ \mathbf{L}^{\text{in}} = \mathbf{z}_1^{\text{in}} x + \mathbf{z}_2^{\text{in}}, \mathbf{L}^{\text{out}} = \mathbf{z}_1^{\text{out}} x + \mathbf{z}_2^{\text{out}} \end{array} \right] = 1.$$

**Security.** A simulator  $\text{KE.Sim}$  for the scheme has the following syntax.

- $\text{KE.Sim}(1^\lambda, \text{pp}, \mathbf{L}^{\text{out}})$  takes as inputs a security parameter  $\lambda$ , public parameters  $\text{pp}$ , and a long label  $\mathbf{L}^{\text{out}} \in \mathcal{L}^{\ell'}$ . It outputs a simulated short label  $\tilde{\mathbf{L}}^{\text{in}} \in \mathcal{L}^\ell$  and a simulated garbled table  $\tilde{\text{tb}}$ .

The scheme is secure if there exists an efficient simulator  $\text{KE.Sim}$  such that for all polynomial  $\ell' = \ell'(\lambda)$ , sequence of key pairs  $\{\mathbf{z}_{1,\lambda}^{\text{out}}, \mathbf{z}_{2,\lambda}^{\text{out}}\}_\lambda$  where  $\mathbf{z}_{1,\lambda}^{\text{out}}, \mathbf{z}_{2,\lambda}^{\text{out}} \in \mathcal{L}^{\ell'}$ , and sequence of inputs  $\{x_\lambda\}_\lambda$  where  $x_\lambda \in \mathcal{I}$ , the following indistinguishability holds. (We suppress the index  $\lambda$  below.)

$$\left\{ \text{pp}, \text{KE.Sim}(1^\lambda, \text{pp}, \mathbf{L}^{\text{out}}) \right\} \approx_c \left\{ \text{pp}, \mathbf{L}^{\text{in}}, \text{tb} \right\} \left| \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda), \mathbf{z}_1^{\text{in}}, \mathbf{z}_2^{\text{in}} \leftarrow \text{KE.KeyGen}^{\text{PP}}(1^\lambda, 1^\ell), \\ \text{tb} \leftarrow \text{KE.Garble}^{\text{PP}}(\mathbf{z}_1^{\text{out}}, \mathbf{z}_2^{\text{out}}, \mathbf{z}_1^{\text{in}}, \mathbf{z}_2^{\text{in}}), \\ \mathbf{L}^{\text{in}} = \mathbf{z}_1^{\text{in}} x + \mathbf{z}_2^{\text{in}}, \mathbf{L}^{\text{out}} = \mathbf{z}_1^{\text{out}} x + \mathbf{z}_2^{\text{out}}. \end{array} \right.$$

## 3 Linearly Homomorphic Encryption

### 3.1 Definition of Basic LHE

We first define a very simple base scheme that creates noisy ciphertexts. Decryption doesn't try to remove the noise, and simply recovers the encrypted message with noise. The scheme allows evaluating linear functions homomorphically over ciphertexts, which increases the level/magnitude of noise in the ciphertexts.

The base scheme can be instantiated under either the learning with error (LWE) assumption or the decisional composite residuosity (DCR) assumption (shown in the full version). We will then implement another scheme on top of a base instantiation that's tailored to the needs of our application.



**Definition 3** (noisy linearly homomorphic encryption). *A noisy linearly homomorphic encryption scheme consists of five efficient algorithms, and is associated with two exponentially bounded functions in the security parameter  $\lambda$ ,  $B_e(\lambda), B_s(\lambda) \leq 2^{\text{poly}(\lambda)}$ .*

- **Setup**( $1^\lambda, 1^\Psi, \text{param}$ ) takes as inputs a security parameter  $\lambda$ , an upper bound  $\Psi \in \mathbb{N}$  on the dimensions of message vectors to be encrypted, and additional parameters **param**. It outputs public parameters **pp**, which defines a key space  $\mathcal{S} = \mathbb{Z}^{\ell_s}$ , a ciphertext space  $\mathcal{E}$ , and a message modulus  $P$ , that satisfy certain properties specified by **param**.
- By default, the rest of the algorithms have random access to **pp**, and receive as inputs  $1^\lambda, \text{param}$  in addition to other inputs, i.e., we use the simplified notation  $X(x_1, x_2, \dots)$  to mean  $X^{\text{pp}}(1^\lambda, \text{param}, x_1, x_2, \dots)$ .
- **KeyGen**( $1^{\ell_s}$ ) takes the key dimension  $\ell_s$  (specified by **pp**) as input, and outputs a key  $s \in \mathbb{Z}^{\ell_s}$ , satisfying that  $|s|_\infty < B_s(\lambda)$ .
- **Enc**( $s, \mathbf{m}$ ) takes as inputs a key  $s \in \mathbb{Z}^{\ell_s}$ , and a message vector  $\mathbf{m} \in \mathbb{Z}^\psi$  of dimension  $\psi \leq \Psi$ . It outputs a ciphertext  $\text{ct} \in \mathcal{E}$ .
- **Dec**( $s, \text{ct}$ ) takes as inputs a key  $s \in \mathbb{Z}^{\ell_s}$ , and a ciphertext  $\text{ct} \in \mathcal{E}$ . It outputs a (noisy) message vector  $\mathbf{m}' \in \mathbb{Z}_P^\psi$  of dimension  $\psi \leq \Psi$ , or the symbol  $\perp$  in case of a decryption error.
- **Eval**( $f, \{\text{ct}_i\}$ ) takes as input a linear function  $f$  specified by  $d$  integer coefficients i.e.,  $f(x_1, \dots, x_d) = \sum_{i \in [d]} a_i x_i$ , and  $d$  ciphertexts  $\{\text{ct}_i\}_{i \in [d]}$ . It outputs an evaluated ciphertext  $\text{ct}_f \in \mathcal{E}$ .

(If  $\text{ct}_i$  encrypts a message vector  $\mathbf{m}_i \in \mathbb{Z}_P^\psi$  of dimension  $\psi \leq \Psi$ , under a key  $s_i$ ,  $\text{ct}_f$  should encrypt the vector  $\mathbf{m}_f = f(\mathbf{m}_1, \dots, \mathbf{m}_d)$ , evaluated coordinate-wise over  $\mathbb{Z}_P$ , under the key  $s_f = f(s_1, \dots, s_d)$ , evaluated over  $\mathbb{Z}$ .)

**Correctness w.r.t.  $B_e$ .** *The scheme is correct if for all  $\lambda, \Psi \in \mathbb{N}$ , **param**,  $\text{pp} \in \text{Supp}(\text{Setup}(1^\lambda, 1^\Psi, \text{param}))$ ,  $s \in \mathbb{Z}^{\ell_s}$ ,  $\mathbf{m} \in \mathbb{Z}^\psi$  where  $\psi \leq \Psi$ , it holds that*

$$\Pr \left[ \|e\|_\infty \leq B_e \mid \begin{array}{l} \text{ct} \leftarrow \text{Enc}(s, \mathbf{m}), \mathbf{m}' = \text{Dec}(s, \text{ct}), \\ e = \mathbf{m}' - \mathbf{m} \pmod{P} \end{array} \right] = 1,$$

where we calculate the infinity norm  $\|\cdot\|_\infty$  of  $e \in \mathbb{Z}_P^\psi$  by identifying it as an integer vector over  $[-P/2, P/2]^\psi$ .

**One-time Security.** *The scheme is (one-time) secure if for all polynomial  $\Psi = \Psi(\lambda)$ , sequence of parameters  $\{\text{param}_\lambda\}_\lambda$  each of bit length  $|\text{param}_\lambda| \leq \text{poly}(\lambda)$ , and sequence of integer message vectors  $\{\mathbf{m}_{1,\lambda}, \mathbf{m}_{2,\lambda}\}_\lambda$  where  $\mathbf{m}_{1,\lambda}, \mathbf{m}_{2,\lambda} \in \mathbb{Z}^{\psi_\lambda}$  of dimension  $\psi_\lambda \leq \Psi(\lambda)$ ,  $\|\mathbf{m}_{i,\lambda}\|_\infty \leq 2^{\text{poly}(\lambda)}$ , the following indistinguishability holds. (We suppress the index  $\lambda$  below.)*

$$\{\text{ct}_1, \text{pp}\} \approx_c \{\text{ct}_2, \text{pp}\} \cdot \left| \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda, 1^\Psi, \text{param}), \\ s \leftarrow \text{KeyGen}(1^{\ell_s}), \text{ct}_i \leftarrow \text{Enc}(s, \mathbf{m}_i), \end{array} \right.$$

Below we define two additional properties satisfied by our base instantiations under either the LWE or the DCR assumption.

**Definition 4** (linear homomorphism). *A LHE scheme (per Definition 3) has linear homomorphism if for all linear function  $f$  specified by  $d$  integer coefficients, for all  $\lambda, \Psi \in \mathbb{N}$ ,  $\text{param}, \text{pp} \in \text{Supp}(\text{Setup}(1^\lambda, 1^\Psi, \text{param}))$ ,  $s_i \in \mathbb{Z}^{\ell_s}$  for each  $i \in [d]$ , and  $\text{ct}_i \in \mathcal{E}$  for each  $i \in [d]$ , such that,  $\text{Dec}(s_i, \text{ct}_i)$  outputs  $\mathbf{m}_i \neq \perp$  and all  $\mathbf{m}_i$  have the same dimension  $\psi \leq \Psi$ , then the following holds:*

$$\Pr \left[ \begin{array}{l} \text{Dec}(s_f, \text{ct}_f) \\ = f(\{\mathbf{m}_i\}) \pmod{P} \end{array} \middle| \begin{array}{l} \mathbf{m}_i = \text{Dec}(s_i, \text{ct}_i), \text{ct}_f \leftarrow \text{Eval}(f, \{\text{ct}_i\}), \\ s_f = f(\{s_i\}) \text{ (over } \mathbb{Z}) \end{array} \right] = 1.$$

**Definition 5** (statistical closeness). *A LHE scheme (per Definition 3) has statistical closeness if for all  $\lambda, \Psi \in \mathbb{N}$ ,  $\text{param}, \text{pp} \in \text{Supp}(\text{Setup}(1^\lambda, 1^\Psi, \text{param}))$ ,  $s \in \mathbb{Z}^{\ell_s}$ , and any two distributions  $D_1, D_2$  of ciphertexts over  $\mathcal{E}$  such that, for all  $i \in \{1, 2\}$ ,  $\Pr[\text{Dec}(s, \text{ct}_i) \neq \perp \mid \text{ct}_i \leftarrow D_i] = 1$ , the following holds:*

$$\Delta_{\text{SD}}(\text{ct}_1, \text{ct}_2) = \Delta_{\text{SD}}(\text{Dec}(s, \text{ct}_1), \text{Dec}(s, \text{ct}_2)) \mid \text{ct}_i \leftarrow D_i, .$$

### 3.2 A Construction of Special-Purpose LHE

We next construct a special-purpose LHE scheme  $\overline{\text{lhe}}$  using a basic LHE scheme  $\text{lhe}$  defined in the previous subsection as a black-box. The special-purpose LHE  $\overline{\text{lhe}}$  is tailored to the needs of our garbling construction in the following ways:

- **ARBITRARY MESSAGE SPACE:** Note that the message space  $\mathbb{Z}_P$  of the basic LHE scheme is specified by the public parameter  $\text{pp}$  during setup time, and may not match domain, e.g.  $\mathbb{Z}_p$  of the computation to be garbled. (For example, in the DCR instantiation,  $P = N^r$ , where  $N$  is a randomly sampled RSA modulus). In  $\overline{\text{lhe}}$ , the setup algorithm  $\overline{\text{Setup}}$  takes an arbitrary modulus  $p$  as an additional parameter, and sets up public parameters  $\overline{\text{pp}}$  with exactly  $\mathbb{Z}_p$  as the message space. In the construction,  $\overline{\text{Setup}}$  invokes the basic setup algorithm  $\text{Setup}$  and makes sure that the basic modulus  $P$  is sufficiently large.  $\overline{\text{lhe}}$  then embeds the actual message space  $\mathbb{Z}_p$  in  $\mathbb{Z}_P$ .
- **EXACT DECRYPTION:**  $\text{lhe}$  decryption produces noisy message, where the noise is bounded by  $B_e$ , while  $\overline{\text{lhe}}$  decryption produces the *exact* message.
- **SPECIAL-PURPOSE LINEAR HOMOMORPHISM, AND NOISE SMUDGING:** Relying on the linear homomorphism and statistical closeness of  $\text{lhe}$  (Definition 4, Definition 5), we show in Lemma 2 and Lemma 1 that  $\overline{\text{lhe}}$  satisfies properties tailored for our construction of garbling schemes. Roughly speaking, it allows evaluating simple linear functions, e.g.,  $f(x_1, x_2) = yx_1 + x_2$ , and  $f'(x_{res}, x_1) = x_{res} - yx_1$ , and we can smudge the noise in a noisy ciphertext by homomorphically adding an encryption of the smudging noise.

**Construction 1** (LHE for  $\mathbb{Z}_p$ ). We construct the special purpose scheme  $\overline{\text{lhe}}$  on top of a basic scheme  $\text{lhe}$  (instantiated under either LWE or DCR in the full version.) Let  $B_e = B_e(\lambda) \leq 2^{\text{poly}(\lambda)}$  be the fixed noise bound for  $\text{lhe}$  guaranteed by its correctness (Definition 3).

–  $\overline{\text{Setup}}(1^\lambda, 1^\Psi, p, B_{\max})$  takes as input an arbitrary message modulus  $p \in \mathbb{N}$ , and an upper bound  $B_{\max} \in \mathbb{N}$  on noise levels in ciphertexts, and proceeds in the following steps:

- Set  $B_{\text{msg}} = 2p \cdot \max(B_{\max}, B_e)$ , and run  $\text{Setup}(1^\lambda, 1^\Psi, \text{param} = B_{\text{msg}})$  to obtain  $\text{pp}$ , which specifies a key dimension  $\ell_s$ , a ciphertext space  $\mathcal{E}$ , and a message modulus  $P$ .

Our instantiations of  $\overline{\text{lhe}}$  guarantee that  $P \geq B_{\text{msg}}$ , and  $\ell_s$  is polynomial in  $\lambda$  and  $\log B_{\text{msg}}$ , independent of the maximal message dimension  $\Psi$ .

- Set a scaling factor  $\Delta = \lfloor P/p \rfloor$ , and output  $\overline{\text{pp}} = (\text{pp}, \Delta)$ , which specifies a key space  $\mathcal{S} = \mathbb{Z}^{\ell_s}$ , a ciphertext space  $\mathcal{E}$ , and a message modulus  $p$ .

Note: *By our setting of  $B_{\text{msg}}$ , and the guarantee that  $P \geq B_{\text{msg}}$ , we have*

$$\Delta \geq \lfloor B_{\text{msg}}/p \rfloor = 2 \max(B_{\max}, B_e) . \quad (3)$$

–  $\overline{\text{KeyGen}}(1^{\ell_s})$  directly runs  $s \leftarrow \text{KeyGen}(1^{\ell_s})$ , and outputs  $s$ .

–  $\overline{\text{Enc}}(s, \mathbf{m})$  takes as input a secret key  $s$  and a message vector  $\mathbf{m} \in \mathbb{Z}^\psi$ . It computes  $\mathbf{m}' = (\mathbf{m} \bmod p) \cdot \Delta \in \mathbb{Z}_p^\psi$ , and outputs  $\text{ct} \leftarrow \text{Enc}(s, \mathbf{m}')$ . Note: *The one-time security of  $\overline{\text{lhe}}$  follows directly from that of  $\text{lhe}$ .*

–  $\overline{\text{Dec}}(s, \text{ct})$  first runs  $\mathbf{m}' = \text{Dec}(s, \text{ct})$  to recover  $\mathbf{m}' \in \mathbb{Z}_p^\psi$ , and then computes  $\mathbf{m}_p = \lfloor \mathbf{m}'/\Delta \rfloor$  to recover  $\mathbf{m}_p \in \mathbb{Z}_p^\psi$ . It outputs  $\mathbf{m}_p$ .

Note: *By the correctness of  $\text{lhe}$ , we have*

$$\text{Dec}(s, \overline{\text{Enc}}(s, \mathbf{m})) = \text{Dec}(s, \text{Enc}(s, \mathbf{m}_p \cdot \Delta)) = \mathbf{m}_p \cdot \Delta + \mathbf{e} \in \mathbb{Z}_p^\psi,$$

for some noise vector  $\mathbf{e}$  such that  $\|\mathbf{e}\|_\infty \leq B_e$ . As noted in Eq. (3), we have  $\Delta \geq 2B_e$ . Hence rounding by  $\Delta$  recovers the correct message  $\mathbf{m}_p \in \mathbb{Z}_p^\psi$  exactly, i.e., the construction has correctness with noise bound  $\overline{B}_e = 0$ .

–  $\overline{\text{Eval}}(f, \{\text{ct}_i\})$  directly runs  $\text{ct}_f \leftarrow \text{Eval}(f, \{\text{ct}_i\})$  and outputs  $\text{ct}_f$ . Note:  $\overline{\text{Eval}} \equiv \text{Eval}$ , hence it can operate on ciphertexts of both  $\overline{\text{lhe}}$  and  $\text{lhe}$ .

Next, relying on the linear homomorphism of  $\text{lhe}$ , we show that  $\overline{\text{lhe}}$  satisfies linear homomorphism w.r.t. linear functions of the form  $f(x_1, x_2) = yx_1 + x_2$ , which suffices for our garbling constructions.

Lemma 1 shows how to “smudge” the noises in an *evaluated*  $\overline{\text{lhe}}$  ciphertext  $\text{ct}_R$  by homomorphically adding a fresh  $\text{lhe}$  encryption  $\text{ct}_e$  of a smudging noise vector  $\mathbf{e}$  to it, as  $\text{ct}'_R = \overline{\text{Eval}}(+, \text{ct}_R, \text{ct}_e)$ . As long as the smudging noise is large enough, the result  $\text{ct}'_R$  is statistically close to homomorphically adding  $\text{ct}_e$  to a *fresh*  $\overline{\text{lhe}}$  ciphertext  $\text{ct}_2$ , as  $\text{ct}'_2 = \overline{\text{Eval}}(+, \text{ct}_2, \text{ct}_e)$ .

Lemma 2 shows how to set the noise upper bound  $B_{\max}$  during  $\overline{\text{Setup}}$  so that evaluated  $\overline{\text{lhe}}$  ciphertexts can still be decrypted exactly. We prove the lemmas in the full version.

**Lemma 1** (noise smudging). *Suppose the underlying LHE scheme  $\text{lhe}$  in Construction 1 satisfies linear homomorphism (Definition 4) and statistical closeness (Definition 5). For all  $\lambda, \Psi, p, B_{\max}, \alpha_1 \in \mathbb{N}$ , set the smudging noise level to*

$$\alpha_2 = \lambda^{\omega(1)} \max(p, B_e, \alpha_1)^2 .$$

For any  $\text{pp} \in \text{Supp}(\overline{\text{Setup}}(1^\lambda, 1^\Psi, p, B_{\max}))$ ,  $s_1, s_2 \in \mathbb{Z}^{\ell_s}$ ,  $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^\psi$  where  $\psi \leq \Psi$ , and function  $f(x_{res}, x_1) = x_{res} - yx_1$ , where  $|y| < p$ , the following two ciphertexts are statistically close, i.e.,  $\Delta_{\text{SD}}(\text{ct}'_2, \text{ct}'_R) \leq \text{negl}(\lambda)$ .

SAMPLING  $\text{ct}'_2$ :

- generate fresh ciphertext  $\text{ct}_2 \leftarrow \overline{\text{Enc}}(s_2, \mathbf{m}_2)$ .
- sample noise  $\mathbf{e} \leftarrow [-\alpha_2, \alpha_2]^\psi$ , and encrypt it using key  $\mathbf{0}$ ,  $\text{ct}_e \leftarrow \text{Enc}(\mathbf{0}, \mathbf{e})$ .
- smudge noise in  $\text{ct}_2$  via  $\text{ct}'_2 \leftarrow \overline{\text{Eval}}(+, \text{ct}_2, \text{ct}_e)$ .

SAMPLING  $\text{ct}'_R$ :

- generate fresh ciphertext  $\text{ct}_1 \leftarrow \overline{\text{Enc}}(s_1, \mathbf{m}_1)$ .
- sample noise  $\mathbf{e}_1 \leftarrow [-\alpha_1, \alpha_1]^\psi$ , and encrypt it using key  $\mathbf{0}$ ,  $\text{ct}_{e,1} \leftarrow \text{Enc}(\mathbf{0}, \mathbf{e}_1)$ .
- generate additionally noisy ciphertext  $\text{ct}'_1 \leftarrow \overline{\text{Eval}}(+, \text{ct}_1, \text{ct}_{e,1})$ .
- generate fresh ciphertext  $\text{ct}_{res} \leftarrow \overline{\text{Enc}}(s_{res}, \mathbf{m}_{res})$ , where  $s_{res} = ys_1 + s_2$ , and  $\mathbf{m}_{res} = y\mathbf{m}_1 + \mathbf{m}_2 \pmod p$ .
- homomorphically evaluate  $f(x_{res}, x_1) = x_{res} - yx_1$  to obtain  $\text{ct}_R \leftarrow \overline{\text{Eval}}(f, \text{ct}_{res}, \text{ct}'_1)$ .
- smudge noise in  $\text{ct}_R$  via  $\text{ct}'_R \leftarrow \overline{\text{Eval}}(+, \text{ct}_R, \text{ct}_e)$ , using the same  $\text{ct}_e$  as above.

THE SIMPLER CASE: The statistical closeness also holds when  $\alpha_1 = 0$  and  $\text{ct}'_R$  is generated using  $\text{ct}_1$  directly, instead of  $\text{ct}'_1$ .

**Lemma 2** (homomorphic evaluation). *Suppose the underlying LHE scheme  $\text{lhe}$  in Construction 1 satisfies linear homomorphism (Definition 4). For all  $\lambda, \Psi, p, \alpha_1, \alpha_2 \in \mathbb{N}$ , if the maximal noise level is set sufficient large*

$$B_{\max} \geq p(p + 1 + \alpha + 2B_e), \quad \alpha = \max(\alpha_1, \alpha_2)$$

then for all  $\overline{\text{pp}} \in \text{Supp}(\overline{\text{Setup}}(1^\lambda, 1^\Psi, p, B_{\max}))$ ,  $s_1, s_2 \in \mathbb{Z}^{\ell_s}$ ,  $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^\psi$  where  $\psi \leq \Psi$ , homomorphic evaluation of functions of form  $f(x_1, x_2) = yx_1 + x_2$  where  $|y| < p$  on additionally noisy ciphertexts yields correct decryption:

$$\Pr \left[ \begin{array}{l} \overline{\text{Dec}}(ys_1 + s_2, \text{ct}_{res}) \\ = y\mathbf{m}_1 + \mathbf{m}_2 \pmod p \end{array} \middle| \begin{array}{l} \forall i \in \{1, 2\}, \mathbf{e}_i \leftarrow [-\alpha_i, \alpha_i]^\psi, \text{ct}_{e,i} \leftarrow \text{Enc}(\mathbf{0}, \mathbf{e}_i), \\ \text{ct}_i \leftarrow \overline{\text{Enc}}(s_i, \mathbf{m}_i), \text{ct}'_i \leftarrow \overline{\text{Eval}}(+, \text{ct}_i, \text{ct}_{e,i}) \\ \text{ct}_{res} \leftarrow \overline{\text{Eval}}(f, \text{ct}'_1, \text{ct}'_2) \end{array} \right] = 1.$$

THE SIMPLER CASE: The above also holds when  $\alpha_1 = 0$  and  $\text{ct}'_{res}$  is generated using  $\text{ct}_1$ , instead of  $\text{ct}'_1$ .

## 4 Key Extension for Bounded Integer Computation

In this section, we construct the key-extension gadget for  $B$ -bounded integer computation. Our starting point is the following observation: A  $B$ -bounded computation can be “embedded” in modulo- $p$  computation as long as  $p > 2B$ :

$$(C, x) \text{ is } B\text{-bounded} \quad \wedge \quad p > 2B \quad \implies \quad C(x) \text{ over } \mathbb{Z} = C(x) \pmod p.$$

### Setup Algorithm of Bounded Integer Garbling

**Parameters and Tools:** The computation is  $B$ -bounded. The construction uses the scheme  $\overline{\text{lhe}}$  from Construction 1, which is associated with a bound  $B_s$  on the infinity norm of LHE keys sampled by  $\overline{\text{lhe.KeyGen}}$ , and a bound  $B_e$  on the decryption noise of the scheme  $\overline{\text{lhe}}$  underlying  $\overline{\text{lhe}}$ . All of  $B$ ,  $B_s$ , and  $B_e$  are bounded by  $2^{\text{poly}(\lambda)}$ .

$\text{Setup}(1^\lambda)$  invokes the setup algorithm of the  $\overline{\text{lhe}}$  scheme

$$\overline{\text{pp}} \leftarrow \overline{\text{lhe.Setup}}(1^\lambda, 1^\Psi, p, B_{\max}),$$

and outputs  $\text{pp} = (\overline{\text{pp}}, \ell)$ , where the parameters are set as below.

- Parameters of the  $\overline{\text{lhe}}$  scheme (with key dimension  $\ell_s = \text{poly}(\lambda, \log B_{\max})$ ):

message modulus	$p = \lambda^{\omega(1)} B \cdot B_s$	(4)
-----------------	---------------------------------------	-----

smudging noise level	$\alpha = \lambda^{\omega(1)} \max(p, B_e)^2$	(5)
----------------------	---	-----

maximal noise level	$B_{\max} = p(p + 1 + \alpha + 2B_e)$
---------------------	---------------------------------------

message dimension bound	$\Psi = 2(\ell_s + 1) = 2\ell$ .
-------------------------	----------------------------------

- The dimension of keys/labels of the key extension gadget is set to  $\ell = \ell_s + 1$ .

**Fig. 4.** Setup for bounded integer garbling.

Therefore, we can directly use the (information theoretic) arithmetic operation gadget for ring  $\mathbb{Z}_p$  from AIK (recalled in the full version). What remains is to design a key-extension gadget for  $\mathbb{Z}_p$ , i.e., a mechanism that enables expanding a short label  $\mathbf{ax} + \mathbf{b} \bmod p$  to an arbitrarily long label  $\mathbf{cx} + \mathbf{d} \bmod p$ .

As shown in this section, the fact that every intermediate values  $x$  is bounded tremendously simplifies the key extension gadget, especially if it is compared with the key extension gadget for modular computation in the full version.

#### 4.1 The Setup Algorithm

Our key extension gadget for bounded integer uses the special-purpose LHE scheme  $\overline{\text{lhe}}$  in Construction 1. The parameters of the LHE scheme is setup once by the Setup algorithm of the entire garbling scheme, as shown in Fig. 4, and is shared by all invocation of gadgets when garbling an arithmetic circuit.

We emphasize that the Setup algorithm depends only on the security parameter and the integer bound  $B$ . It's independent of any parameters (e.g., maximal size, fan-out, depth) of the circuit to be garbled later. As such, the public parameter  $\text{pp}$  is generated once and re-used for garbling many poly-sized circuits.

#### 4.2 Length-Doubling Key Extension

We present the construction in two steps:

**Step 1: Length-doubling.** In Construction 2, we present a basic *length-doubling* key extension gadget, that is, at evaluation time, given a label  $\mathbf{z}_1^{in}x + \mathbf{z}_2^{in}$  of dimension  $\ell$  produces a label  $\mathbf{z}_1^{out}x + \mathbf{z}_2^{out}$  of dimension  $2\ell$ . This construction already contains our main idea.

**Step 2: Arbitrary Expansion.** Next, we present a generic transformation (in the full version) that converts a *length-doubling* key extension gadget, to a full-fledged key extension gadget that produces an output-wire label of arbitrary polynomial dimension  $\ell' > \ell$ . At a high-level, the transformation recursively calls the *length-doubling* key extension gadget in a tree fashion till the desired output-wire label dimension  $\ell'$  is reached.

**Construction 2** (length-doubling key extension for bounded integers). The algorithms below uses the parameters,  $p, \alpha, B_{\max}, \Psi$ , specified in Setup (Fig. 4), and have random access to the public parameters  $\mathbf{pp}$ , which contains the public parameter  $\overline{\mathbf{pp}}$  of the LHE scheme  $\overline{\text{lhe}}$  and the key dimension  $\ell$ .

- $\text{KE.KeyGen}^{\text{PP}}(1^\lambda, 1^\ell)$ : Generate a  $\overline{\text{lhe}}$  secret key  $\mathbf{s}_1 \leftarrow \overline{\text{lhe.KeyGen}}(1^{\ell_s})$ , which is an integer vector in  $\mathbb{Z}^{\ell_s}$  with  $\|\mathbf{s}_1\|_\infty \leq B_s$ . Output input-wire keys  $\mathbf{z}_1, \mathbf{z}_2$ :

$$\mathbf{z}_1^{in} = (\mathbf{s}_1, 1), \quad \mathbf{z}_2^{in} = (r\mathbf{s}_1 + \mathbf{s}_2, r) \quad (\text{over } \mathbb{Z}),$$

where  $r \leftarrow [-B_{\text{smdg}}, B_{\text{smdg}}]$  and  $\mathbf{s}_2 \leftarrow [-B'_{\text{smdg}}, B'_{\text{smdg}}]^{\ell_s}$ , with  $B_{\text{smdg}} = \lambda^{\omega(1)}B$  and  $B'_{\text{smdg}} = \lambda^{\omega(1)}B_{\text{smdg}}B_s < p/4$  (the inequality can be satisfied because the message modulus  $p$  is set sufficient large; see Equation (4)).

Note: We make a few observations: *i*) the input-wire keys are  $p$  bounded, that is, they belong to the label/key space  $\mathbf{z}_1^{in}, \mathbf{z}_2^{in} \in \mathbb{Z}_p$  as the definition requires, and *ii*) a label for  $x$  equals

$$\begin{aligned} \mathbf{L}^{in} = \mathbf{z}_1^{in}x + \mathbf{z}_2^{in} &= (\mathbf{s}_1(x+r) + \mathbf{s}_2, x+r) \bmod p \\ &= \underbrace{(\mathbf{s}_1(x+r) + \mathbf{s}_2)}_{\mathbf{s}_{res}} \underbrace{(x+r)}_y \text{ over } \mathbb{Z} \end{aligned}$$

The last equality holds because the magnitude of entries of  $\mathbf{s}_{res}$  and  $y$  do not exceed  $p/2$ . The fact that the labels are effectively computed over the integers is crucial for decoding later, and this crucially relies on the fact that values  $x$  are  $B$ -bounded and that  $p$  can be set sufficiently larger than  $B$ .

- $\text{KE.Garble}^{\text{PP}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in})$ : First recover  $\mathbf{s}_1, \mathbf{s}_2, r$  from the input-wire keys. Then encrypt  $\mathbf{z}_1^{out}$  and  $\mathbf{z}_2^{out} = \mathbf{z}_2^{out} - r\mathbf{z}_1^{out}$  using  $\overline{\text{lhe}}$  under keys  $\mathbf{s}_1, \mathbf{s}_2$  respectively. This is possible because  $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$  has dimension  $2\ell \leq \Psi$ , as set in Fig. 4, and any integer vector of dimension  $\ell_s$ , e.g.  $\mathbf{s}_2$ , can be used as a secret key for  $\overline{\text{lhe}}$ .

$$\text{ct}_1 \leftarrow \overline{\text{lhe.Enc}}(\mathbf{s}_1, \mathbf{z}_1^{out}), \quad \text{ct}_2 \leftarrow \overline{\text{lhe.Enc}}(\mathbf{s}_2, \mathbf{z}_2^{out}).$$

Finally, add a smudging noise of magnitude  $\alpha$  (set in Equation (5)) to  $\text{ct}_2$  to obtain  $\text{ct}'_2$ , and output garbled table  $\text{tb} = (\text{ct}_1, \text{ct}'_2)$ .

$$\mathbf{e} \leftarrow [-\alpha, \alpha]^{\ell'} \quad , \quad \text{ct}_e \leftarrow \overline{\text{lhe.Enc}}(0, \mathbf{e}) \quad , \quad \text{ct}'_2 \leftarrow \overline{\text{lhe.Eval}}(+, \text{ct}_2, \text{ct}_e) \quad .$$

- $\text{KE.Dec}^{\text{PP}}(\mathbf{L}^{\text{in}}, \text{tb} = (\text{ct}_1, \text{ct}'_2))$  Treat  $\mathbf{L}^{\text{in}}$  as an integer vector and parse it as  $\mathbf{L}^{\text{in}} = (\mathbf{s}_{\text{res}}, y)$ , where  $\mathbf{s}_{\text{res}} \in \mathbb{Z}^{\ell_s}$ ,  $y \in \mathbb{Z}$ . Homomorphically evaluate the linear function  $f(x_1, x_2) = yx_1 + x_2$  over  $\text{ct}_1$  and  $\text{ct}'_2$ , decrypt the output ciphertext to obtain  $\mathbf{m}_{\text{res}}$ , and output  $\mathbf{L}^{\text{out}} = \mathbf{m}_{\text{res}}$  as the output-wire label:

$$\text{ct}'_{\text{res}} \leftarrow \overline{\text{he.Eval}}(f, \text{ct}_1, \text{ct}'_2), \quad \mathbf{L}^{\text{out}} = \mathbf{m}_{\text{res}} = \overline{\text{he.Dec}}(\mathbf{s}_{\text{res}}, \text{ct}'_{\text{res}}).$$

**Correctness.** We show that the above scheme is *correct*, which requires that given a correctly generated input-wire label  $\mathbf{L}^{\text{in}} = \mathbf{z}_1^{\text{in}}x + \mathbf{z}_2^{\text{in}} \pmod{p}$  and garbled table  $\text{tb}$ , the decoding algorithm  $\text{KE.Dec}$  recovers the correct output-wire label  $\mathbf{L}^{\text{out}} = \mathbf{z}_1^{\text{out}}x + \mathbf{z}_2^{\text{out}} \pmod{p}$ . As we analyzed above  $\mathbf{L}^{\text{in}} = (\mathbf{s}_{\text{res}}, y)$  where  $\mathbf{s}_{\text{res}} = \mathbf{s}_1y + \mathbf{s}_2$  and  $y = x + r$  are computed over the integers. By construction,  $\text{KE.Dec}$  uses  $\mathbf{s}_{\text{res}}$  as the secret key to decrypt the  $\overline{\text{he}}$  ciphertext  $\text{ct}'_{\text{res}}$ , where  $\text{ct}'_{\text{res}}$  is the output ciphertext obtained by homomorphically evaluating  $f(x_1, x_2) = yx_1 + x_2$  over  $\text{ct}_1$  and  $\text{ct}_2$  encrypting  $\mathbf{z}_1^{\text{out}}$  and  $\mathbf{z}_2^{\text{out}}$  respectively. By the special-purpose linear homomorphism of  $\overline{\text{he}}$ , namely Lemma 2 (the simpler case),  $\text{ct}'_{\text{res}}$  can be decrypted using secret key  $f(\mathbf{s}_1, \mathbf{s}_2) = \mathbf{s}_1y + \mathbf{s}_2$  computed over the integers, which is exactly  $\mathbf{s}_{\text{res}}$ . Therefore,

$$\begin{aligned} \mathbf{m}_{\text{res}} &= \overline{\text{he.Dec}}(\mathbf{s}_{\text{res}} = (\mathbf{s}_1y + \mathbf{s}_2), \text{ct}'_{\text{res}}) = (y\mathbf{z}_1^{\text{out}} + \mathbf{z}_2^{\text{out}}) \pmod{p} \\ &= \underbrace{((x+r)\mathbf{z}_1^{\text{out}})}_{=y} + \underbrace{(\mathbf{z}_2^{\text{out}} - r\mathbf{z}_1^{\text{out}})}_{\mathbf{z}'_2^{\text{out}}} \pmod{p} = \mathbf{z}_1^{\text{out}}x + \mathbf{z}_2^{\text{out}} \pmod{p} = \mathbf{L}^{\text{out}}. \end{aligned}$$

In order to invoke Lemma 2, we still need to verify that the prerequisite  $B_{\max} \geq p(p+1+\alpha+2B_e)$  is indeed satisfied. This is the case as set by Setup in Fig. 4.

The security proof of the scheme is deferred to the full version.

**Lemma 3.** *Construction 2 is secure per Definition 2.*

## 5 Potential for Concrete Efficiency Improvement

In this section, we compare the concrete efficiency of our bounded integer garbling scheme based on the DCR assumption against the scheme of [8] (BMR), which garbles arithmetic circuits in the bounded integer model with free addition and subtraction, and the baseline solution that first converts arithmetic circuits into Boolean circuits and then runs the Boolean garbling scheme of [17] (RR). We defer the analysis and comparison for our mod- $p$  and mixed garbling to the full version. Note that, the concrete efficiency of our construction is not optimized. The calculations and comparisons in this section are only to demonstrate the potential towards more practical garbling schemes, for garbling arithmetic circuits with large domains.

Concretely, for  $B$ -bounded integer garbling, we consider the Paillier modulus  $N$  to have 4096 bits, and the bit length  $\ell = \log(B)$  to be just slightly below 4096, specifically 3808 bits (the setting is described below). We set the statistical security parameter  $\kappa = 80$ .

**Table 3.** Comparison of garbling circuit size for bounded integer computation. The last two lines assume the stronger small exponent assumption.

Scheme	Garbled Table Size	
Ours (per Mult Gate)	$12 \cdot 3 \cdot \log N$	18.0 KB
[17] (per Mult Gate)	$1.5 \cdot 128 \cdot \ell^{1.58}$	10.4 MB
[8] (per Mult Gate)	$2 \cdot 128 \cdot \sum_{i=1}^k (p_i - 1)$	15.0 MB
Ours (per +/− Gate)	$6 \cdot 3 \cdot \log N$	9.0 KB
[17] (per +/− Gate)	$1.5 \cdot 128 \cdot \ell \log \ell$	1.0 MB
[8] (per +/− Gate)	Free	0 b
Ours, Improved (per Mult Gate)	$12 \cdot 2 \cdot \log N$	12.0 KB
Ours, Improved (per +/− Gate)	$6 \cdot 2 \cdot \log N$	6.0 KB

Under the concrete setting, the most efficient Boolean circuit implementation for integer multiplication uses Karatsuba’s method. We conservatively count the number of AND gates (as XOR gates in RR is free) in a multiplication circuit as  $\ell^{1.58}$ , ignoring any hidden constants, and an addition circuit as  $\ell \log \ell$ .

At a high level, the BMR scheme works by decomposing a large  $B$ -bounded integer into its Chinese Remainder Theorem (CRT) representation using the smallest distinct primes ( $p_1 = 2, p_2 = 3, \dots, p_k$ ) whose product exceeds  $B$ . Under the concrete setting, the number of primes is  $k = 394$ .

**Size of Bounded Integer Garbling.** Under standard DCR, our bounded integer garbling significantly improves the garbling size of both addition ( $\sim 100\times$ ) and multiplication ( $\sim 500\times$ ) gates over the Boolean baseline using RR, as shown in Table 3. BMR supports free addition, but multiplication is more expensive than RR.

The formula for our scheme is derived as follows. In our garbling scheme, the garbled table for each multiplication gate consists of 12 ring elements in  $\mathbb{Z}_P$ : according to Fig. 1, the two input wires each have a pair of keys of dimension 4 and 2 (as the label  $x_{j_2} + s$  is available before key extension and does not need to be regenerated). In the DCR instantiation, we set  $P = N^3$ . Because  $\ell = \log B = 3808$ , it satisfies that  $N^2 \geq N^{2^{2\kappa}} B$ , which is how large the values encrypted in Paillier encryption are. Note that this is different from how Setup algorithm (Fig. 4) specifies the modulus  $P$ , because Setup is designed to fit both the DCR and the LWE instantiations. The size of garbling an addition gate is calculated the same way as multiplication, except with key dimensions 2 and 1 for the input wires.

**Computation Efficiency.** In both BMR and RR, the main costs are computing garbled table entries, which are 128-bit AES ciphertexts. Concretely: BMR computes  $2 \cdot \sum_{i=1}^k (p_i - 1) \approx 10^6$  AES ciphertexts for each Mult gate, and has free addition. The Boolean baseline using RR computes  $1.5 \cdot \ell^{1.58} \approx 6.8 \times 10^5$  and  $1.5 \cdot \ell \log \ell \approx 6.8 \times 10^4$  AES ciphertexts for each Mult and Add gate respectively. In our scheme, a garbled table for Mult consists of 12 ring elements in



**Table 4.** Comparison of computation costs for bounded integer garbling.

Scheme	Garbling Computation Cost	
Ours (per Mult Gate)	$12 \cdot \text{lsk} \approx 1.5 \times 10^3$	Mult mod $P$
[17] (per Mult Gate)	$1.5 \cdot \ell^{1.58} \approx 6.8 \times 10^5$	AES calls
[8] (per Mult Gate)	$2 \cdot \sum_{i=1}^k (p_i - 1) \approx 10^6$	AES calls
Ours (per +/- Gate)	$6 \cdot \text{lsk} \approx 7.7 \times 10^2$	Mult mod $P$
[17] (per +/- Gate)	$1.5 \cdot \ell \log \ell \approx 6.8 \times 10^4$	AES calls
[8] (per +/- Gate)	Free	Free

$\mathbb{Z}_P$ , each a Paillier ciphertext of the form  $h^x(1+N)^m$ , for some hard group element  $h$ , secret exponent  $x$ , and message  $m$ . Thanks to algebraic properties of Pailler,  $(1+N)^m$  can be computed cheaply without exponentiation. The main cost comes from raising  $h$  to the exponent  $x$ . Let  $\text{lsk}$  be the bit length of  $x$ . The DCR assumption assumes that  $\text{lsk} = \log N = 4096$ . However, under the “small exponent” assumption as introduced in [1], we can set  $\text{lsk} = 128$ , which improves efficiency in two ways. First, it slightly reduces our garbling sizes, as we can now set the modulus  $P = N^2 \geq N^{2^{\text{lsk}}} 2^{2^\kappa} B$ , as shown in the last two lines of Table 3. More importantly, it significantly improves computational efficiency. Concretely, our scheme computes  $12 \cdot \text{lsk} \approx 1.5 \times 10^3$  and  $6 \cdot \text{lsk} \approx 7.7 \times 10^2$  multiplications mod  $P$  for each Mult and Add gate respectively. A comparison is in Table 4.

**Acknowledgement.** The authors would like to thank the anonymous Eurocrypt reviewers for their valuable and insightful comments.

Huijia Lin and Hanjun Li were supported by NSF grants CNS-1936825 (CAREER), CNS-2026774, a JP Morgan AI research Award, a Cisco research award, and a Simons Collaboration on the Theory of Algorithmic Fairness.



## References

1. Abram, D., Damgård, I., Orlandi, C., Scholl, P.: An algebraic framework for silent preprocessing with trustless setup and active security. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 421–452. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-15985-5\\_15](https://doi.org/10.1007/978-3-031-15985-5_15)
2. Ananth, P., Jain, A., Sahai, A.: Indistinguishability obfuscation for turing machines: constant overhead and amortization. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 252–279. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-319-63715-0\\_9](https://doi.org/10.1007/978-3-319-63715-0_9)
3. Applebaum, B., Avron, J., Brzuska, C.: Arithmetic cryptography: extended abstract. In: Roughgarden, T. (ed.) ITCS 2015, pp. 143–151. ACM (2015). <https://doi.org/10.1145/2688073.2688114>
4. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in  $\text{NC}^0$ . In: 45th FOCS, pp. 166–175. IEEE (2004). <https://doi.org/10.1109/FOCS.2004.20>
5. Applebaum, B., Ishai, Y., Kushilevitz, E.: How to garble arithmetic circuits. In: Ostrovsky, R. (ed.) 52nd FOCS, pp. 120–129. IEEE (2011). <https://doi.org/10.1109/FOCS.2011.40>

6. Applebaum, B., Ishai, Y., Kushilevitz, E., Waters, B.: Encoding functions with constant online rate or how to compress garbled circuits keys. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 166–184. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40084-1\\_10](https://doi.org/10.1007/978-3-642-40084-1_10)
7. Ball, M., Carmer, B., Malkin, T., Rosulek, M., Schimanski, N.: Garbled neural networks are practical. IACR Cryptol. ePrint Arch, 338 (2019)
8. Ball, M., Malkin, T., Rosulek, M.: Garbling gadgets for Boolean and arithmetic circuits. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 565–577. ACM Press (2016). <https://doi.org/10.1145/2976749.2978410>
9. Boneh, D., et al.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 533–556. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-642-55220-5\\_30](https://doi.org/10.1007/978-3-642-55220-5_30)
10. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-44586-2\\_9](https://doi.org/10.1007/3-540-44586-2_9)
11. 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, Part II. LNCS, vol. 8043, pp. 536–553. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40084-1\\_30](https://doi.org/10.1007/978-3-642-40084-1_30)
12. Harvey, D., Van Der Hoeven, J.: Integer multiplication in time  $o(n \log n)$ . Ann. Math. **193**(2), 563–617 (2021)
13. Ishai, Y., Wee, H.: Partial garbling schemes and their applications. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part I. LNCS, vol. 8572, pp. 650–662. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-662-43948-7\\_54](https://doi.org/10.1007/978-3-662-43948-7_54)
14. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-70583-3\\_40](https://doi.org/10.1007/978-3-540-70583-3_40)
15. Li, H., Lin, H., Luo, J.: ABE for circuits with constant-size secret keys and adaptive security. IACR Cryptol. ePrint Arch, 659 (2022). <https://eprint.iacr.org/2022/659>
16. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48910-X\\_16](https://doi.org/10.1007/3-540-48910-X_16)
17. Rosulek, M., Roy, L.: Three halves make a whole? beating the half-gates lower bound for garbled circuits. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 94–124. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84242-0\\_5](https://doi.org/10.1007/978-3-030-84242-0_5)
18. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: 23rd FOCS, pp. 160–164. IEEE (1982). <https://doi.org/10.1109/SFCS.1982.38>
19. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46803-6\\_8](https://doi.org/10.1007/978-3-662-46803-6_8)



# Actively Secure Half-Gates with Minimum Overhead Under Duplex Networks

Hongrui Cui<sup>1</sup> , Xiao Wang<sup>2</sup> , Kang Yang<sup>3</sup>  , and Yu Yu<sup>1,4</sup> 

<sup>1</sup> Shanghai Jiao Tong University, Shanghai, China  
{rickfreeman, yyuu}@sjtu.edu.cn

<sup>2</sup> Northwestern University, Evanston, USA  
wangxiao@cs.northwestern.edu

<sup>3</sup> State Key Laboratory of Cryptology, Beijing, China  
yangk@sklc.org

<sup>4</sup> Shanghai Qi Zhi Institute, Shanghai, China

**Abstract.** Actively secure two-party computation (2PC) is one of the canonical building blocks in modern cryptography. One main goal for designing actively secure 2PC protocols is to reduce the communication overhead, compared to semi-honest 2PC protocols. In this paper, we propose a new actively secure constant-round 2PC protocol with one-way communication of  $2\kappa + 5$  bits per AND gate (for  $\kappa$ -bit computational security and any statistical security), essentially matching the one-way communication of semi-honest half-gates protocol. This is achieved by two new techniques:

1. The recent compression technique by Dittmer et al. (Crypto 2022) shows that a relaxed preprocessing is sufficient for authenticated garbling that does not reveal masked wire values to the garbler. We introduce a new form of authenticated bits and propose a new technique of generating authenticated AND triples to reduce the one-way communication of preprocessing from  $5\rho + 1$  bits to 2 bits per AND gate for  $\rho$ -bit statistical security.
2. Unfortunately, the above compressing technique is only compatible with a less compact authenticated garbled circuit of size  $2\kappa + 3\rho$  bits per AND gate. We designed a new authenticated garbling that does not use information theoretic MACs but rather dual execution without leakage to authenticate wire values in the circuit. This allows us to use a more compact half-gates based authenticated garbled circuit of size  $2\kappa + 1$  bits per AND gate, and meanwhile keep compatible with the compression technique. Our new technique can achieve one-way communication of  $2\kappa + 5$  bits per AND gate.

Our technique of yielding authenticated AND triples can also be used to optimize the two-way communication (i.e., the total communication) by combining it with the authenticated garbled circuits by Dittmer et al., which results in an actively secure 2PC protocol with two-way communication of  $2\kappa + 3\rho + 4$  bits per AND gate.

**Keywords:** Actively secure 2PC · Garbled circuit · Correlated oblivious transfer

# 1 Introduction

Based on garbled circuits (GCs) [44], constant-round secure two-party computation (2PC) has obtained huge practical improvements in recent years in both communication [5, 30, 35, 45] and computation [6, 22, 23]. However, compared to passively secure (a.k.a., semi-honest) 2PC protocols, their actively secure counterparts require significant overhead. Building upon the authenticated garbling framework [29, 36, 37, 42] and, more generally, working in the BMR family [5, 24, 26, 31, 32], the most recent work by Dittmer, Ishai, Lu and Ostrovsky [16] (denoted as DILO hereafter) is able to bring down the communication cost to  $2\kappa + 8\rho + O(1)$  bits per AND gate, where  $\kappa$  and  $\rho$  are the computational and statistical security parameters, respectively.

Although huge progress, there is still a gap between actively secure and passively secure 2PC protocols based on garbled circuits. In particular, the size of a garbled circuit has been recently reduced from  $2\kappa$  bits (half-gates [45]) to  $1.5\kappa$  bits (three-halves [35]) per AND gate, while even the latest authenticated garbling cannot reach the communication efficiency of half-gates. It is possible to close this gap between active and passive security using the GMW compiler [21], and its concrete efficiency was studied in [1]. However, it requires non-black-box use of the underlying garbling scheme and thus requires prohibitive overhead.

Bringing down the cost of authenticated garbling at this stage requires overcoming several challenges. First of all, we need the authenticated GC itself to be as small as the underlying GC construction. This could be achieved for half-gates as Katz et al. [29] (denoted as KRRW hereafter) proposed an authenticated half-gates construction in the two-party setting. However, when it comes to three-halves, there is no known construction. These authenticated GCs are usually generated in some preprocessing model, and thus the second challenge is to instantiate the preprocessing with only *constant additive overhead*. Together with recent works on pseudorandom correlation generators (PCGs) [9–11, 13, 43], Katz et al. [29] can achieve  $O(\kappa)$  bits per AND gate, while Dittmer et al. [16] can achieve  $O(\rho)$  bits per AND gate. However, the latest advancement by Dittmer et al. [16] is not compatible with the optimal authenticated half-gates construction and requires an authenticated GC of size  $2\kappa + 3\rho$  bits per AND gate.

## 1.1 Our Contribution

We make significant progress in closing the communication gap between passive and active GC-based 2PC protocols by proposing a new actively secure 2PC protocol with constant rounds and one-way communication essentially the same as the half-gates 2PC protocol in the semi-honest setting.

1. We manage to securely instantiate the preprocessing phase with  $O(1)$  bits per AND gate. Our starting point is the compression technique by Dittmer et al. [16], who showed that in authenticated garbling, the random masks of the evaluator need not be of full entropy and can be compressed with entropy sublinear to the circuit size. This observation leads to an efficient construction

**Table 1.** Comparing our protocol with prior works in terms of round and communication complexity. Here  $\kappa, \rho$  denote the computational and statistical security parameters instantiated by 128 and 40 respectively. Round complexity is counted in the random COT/VOLE-hybrid model. One-way communication is the greater of the two parties’ communication; two-way communication is the sum of all communication. For the KRRW and HSS protocol we take the bucket size as  $B = 3$ .

Security	2PC	Correlation	Rounds		Communication per AND gate	
			Prep.	Online	one-way (bits)	two-way (bits)
Passive	Half-gates	OT	1	2	$2\kappa$	$2\kappa$
Active	HSS-PCG [25]	OT	8	2	$8\kappa + 11$ (4.04 $\times$ )	$16\kappa + 22$ (8.09 $\times$ )
Active	KRRW-PCG [29]	COT	4	4	$5\kappa + 7$ (2.53 $\times$ )	$8\kappa + 14$ (4.05 $\times$ )
Active	DILO [16]	VOLE	7	2	$2\kappa + 8\rho + 1$ (2.25 $\times$ )	$2\kappa + 8\rho + 5$ (2.27 $\times$ )
Active	This work	COT	8	3	$2\kappa + 5$ ( $\approx 1\times$ )	$4\kappa + 10$ (2.04 $\times$ )
Active	This work+DILO	COT	8	2	$2\kappa + 3\rho + 2$ (1.48 $\times$ )	$2\kappa + 3\rho + 4$ ( $\approx 1.48\times$ )

from vector oblivious linear evaluation (VOLE) to the desired preprocessing functionality. This reduces the communication overhead of preprocessing to  $5\rho + 1$  bits per AND gate. To further reduce their communication, we introduce a new tool called “dual-key authentication”. Intuitively this form of authentication allows two parties to commit to a value that can later be checked against subsequent messages by both parties. Together with a new technique of generating authenticated AND triples from correlated oblivious transfer (COT), we avoid the  $\rho$ -time blow-up of the DILO protocol, and the one-way communication cost is reduced to 2 bits per AND gate.

- As mentioned earlier, the above compression technique is not compatible with KRRW authenticated half-gates; this is because the compression technique requires that the garbler does not learn the masked values since the entropy of wire masks provided by the evaluator is low. We observe that the dual-execution protocol [27, 28] can essentially be used for this purpose, and it is highly compatible with the authenticated garbling technique. In particular, the masked value of each wire is implicitly authenticated by the garbled label. Therefore we can perform two independent executions and check the actual value of each wire against each other. Since every wire is checked, we are able to eliminate the 1-bit leakage in ordinary dual-execution protocols. The overall one-way communication is  $2\kappa + 5$  bits per AND gate.

We note that this is only a partial solution because dual execution requires both parties to send GCs. Under full-duplex networks (e.g., most wired communication) where communication in both directions can happen simultaneously, this effectively imposes no slow down; however, for half-duplex networks (e.g., most wireless communication), it would not be a preferable option. Nevertheless, our preprocessing protocol can be combined with the construction of authenticated garbled circuits by Dittmer et al. [16] to achieve the best two-way communication of  $2\kappa + 3\rho + 4$  bits per AND gate, leading to a 1.53 $\times$  improvement. We provide a detailed comparison in Table 1.

We do not compare our actively secure 2PC protocol with the protocol (denoted by DILOv2) by Dittmer et al. [16] building on doubly authenticated multiplication triples. Compared to DILO, the DILOv2 protocol is less efficient, as DILOv2 requires quasi-linear computational complexity. Moreover, DILOv2 can only generate authenticated triples over  $\mathbb{F}_{2^\rho}$ , while authenticated garbling requires triples over  $\mathbb{F}_2$ . This incurs a  $\rho$ -time overhead when utilizing such triples.

## 2 Preliminaries

### 2.1 Notation

We use  $\kappa$  and  $\rho$  to denote the computational and statistical security parameters, respectively. We use  $\log$  to denote logarithms in base 2. We write  $x \leftarrow S$  to denote sampling  $x$  uniformly at random from a finite set  $S$ . We define  $[a, b) = \{a, \dots, b - 1\}$  and write  $[a, b] = \{a, \dots, b\}$ . We use bold lower-case letters like  $\mathbf{a}$  for column vectors, and bold upper-case letters like  $\mathbf{A}$  for matrices. We let  $a_i$  denote the  $i$ -th component of  $\mathbf{a}$  (with  $a_1$  the first entry). We use  $\{x_i\}_{i \in S}$  to denote the set that consists of all elements with indices in set  $S$ . When the context is clear, we abuse the notation and use  $\{x_i\}$  to denote such a set. For a string  $x$ , we use  $\text{lsb}(x)$  to denote the least significant bit (LSB) and  $\text{msb}(x)$  to denote the most significant bit (MSB).

For an extension field  $\mathbb{F}_{2^\kappa}$  of a binary field  $\mathbb{F}_2$ , we fix some monic, irreducible polynomial  $f(X)$  of degree  $\kappa$  and then write  $\mathbb{F}_{2^\kappa} \cong \mathbb{F}_2[X]/f(X)$ . Thus, every element  $x \in \mathbb{F}_{2^\kappa}$  can be denoted uniquely as  $x = \sum_{i \in [0, \kappa)} x_i \cdot X^i$  with  $x_i \in \mathbb{F}_2$  for all  $i \in [0, \kappa)$ . We could view elements over  $\mathbb{F}_{2^\kappa}$  equivalently as vectors in  $\mathbb{F}_2^\kappa$  or strings in  $\{0, 1\}^\kappa$ , and consider a bit  $x \in \mathbb{F}_2$  as an element in  $\mathbb{F}_{2^\kappa}$ . Depending on the context, we use  $\{0, 1\}^\kappa$ ,  $\mathbb{F}_2^\kappa$  and  $\mathbb{F}_{2^\kappa}$  interchangeably, and thus addition in  $\mathbb{F}_2^\kappa$  and  $\mathbb{F}_{2^\kappa}$  corresponds to XOR in  $\{0, 1\}^\kappa$ . We also define two macros to convert between  $\mathbb{F}_{2^\kappa}$  and  $\mathbb{F}_2^\kappa$ .

- $x \leftarrow \text{B2F}(\mathbf{x})$ : Given  $\mathbf{x} = (x_0, \dots, x_{\kappa-1}) \in \mathbb{F}_2^\kappa$ , output  $x := \sum_{i \in [0, \kappa)} x_i \cdot X^i \in \mathbb{F}_{2^\kappa}$ .
- $\mathbf{x} \leftarrow \text{F2B}(x)$ : Given  $x = \sum_{i \in [0, \kappa)} x_i \cdot X^i \in \mathbb{F}_{2^\kappa}$ , output  $\mathbf{x} = (x_0, \dots, x_{\kappa-1}) \in \mathbb{F}_2^\kappa$ .

A Boolean circuit  $\mathcal{C}$  consists of a list of gates in the form of  $(i, j, k, T)$ , where  $i, j$  are the indices of input wires,  $k$  is the index of output wire and  $T \in \{\oplus, \wedge\}$  is the type of the gate. In the 2PC setting, we use  $\mathcal{I}_A$  (resp.,  $\mathcal{I}_B$ ) to denote the set of circuit-input wire indices corresponding to the input of  $\mathsf{P}_A$  (resp.,  $\mathsf{P}_B$ ). We also use  $\mathcal{W}$  to denote the set of output-wire indices of all AND gates, and  $\mathcal{O}$  to denote the set of circuit-output wire indices in the circuit  $\mathcal{C}$ . We denote by  $\mathcal{C}_{\text{and}}$  the set of all AND gates in the form of  $(i, j, k, T)$ .

Our protocol in the two-party setting is proven secure against static and malicious adversaries in the standard simulation-based security model [12, 20]. We recall the security model, a relaxed equality-check functionality  $\mathcal{F}_{\text{EQ}}$  and the coin-tossing functionality  $\mathcal{F}_{\text{Rand}}$  as well as the summary of the notations and macros used in our protocols in the full version [14].

## 2.2 Information-Theoretic Message Authentication Codes

We use information-theoretic message authentication codes (IT-MACs) [7, 34] to authenticate bits or field elements in  $\mathbb{F}_{2^\kappa}$ . Specifically, let  $\Delta \in \mathbb{F}_{2^\kappa}$  be a *global key*. We adopt  $[x] = (K[x], M[x], x)$  to denote that an element  $x \in \mathbb{F}$  (where  $\mathbb{F} \in \{\mathbb{F}_2, \mathbb{F}_{2^\kappa}\}$ ) known by one party can be authenticated by the other party who holds  $\Delta \in \mathbb{F}_{2^\kappa}$  and a *local key*  $K[x] \in \mathbb{F}_{2^\kappa}$ , where a MAC tag  $M[x] = K[x] + x \cdot \Delta \in \mathbb{F}_{2^\kappa}$  is given to the party holding  $x$ . For a vector  $\mathbf{x} \in \mathbb{F}^\ell$ , we denote by  $[\mathbf{x}] = ([x_1], \dots, [x_\ell])$  a vector of authenticated values. We refer to  $([x], [y], [z])$  with  $z = x \cdot y$  as an authenticated multiplication triple. If  $x, y, z \in \{0, 1\}$ , this tuple is also called authenticated AND triple. For a constant value  $c \in \mathbb{F}_{2^\kappa}$ , it is easy to define  $[c] = (c \cdot \Delta, 0^\kappa, c)$ . It is well-known that IT-MACs are additively homomorphic. That is, given public coefficients  $c_0, c_1, \dots, c_\ell \in \mathbb{F}_{2^\kappa}$ , two parties can *locally* compute  $[y] := c_0 + \sum_{i=1}^{\ell} c_i \cdot [x_i]$ .

When applying IT-MACs into 2PC, secret values are authenticated by either  $P_A$  or  $P_B$ . We use subscripts A and B in authenticated values to distinguish which party ( $P_A$  or  $P_B$ ) holds the secret values. For example,  $[x]_A = (K_B[x], M_A[x], x)$  denotes that  $P_A$  holds  $(x, M_A[x])$  and  $P_B$  holds  $(\Delta_B, K_B[x])$ . In the case that other global keys are used, we explicitly add a subscript to keys and MAC tags. For example, when  $G \in \mathbb{F}_{2^\kappa}$  is used and held by  $P_B$ , we write  $[x]_{A,G} = (K_B[x]_G, M_A[x]_G, x)$  and  $M_A[x]_G = K_B[x]_G + x \cdot G$ . When the context is clear, we will omit the subscripts A and B for the sake of simplicity.

**Batch Opening of Authenticated Values.** In the following, we describe the known procedure [15, 34] to open authenticated values in a batch. Here we always assume that  $P_A$  holds the values and MAC tags, and  $P_B$  holds the global and local keys. In this case, we write  $[x]$  instead of  $[x]_A$ . For the case that  $P_B$  holds the values authenticated by  $P_A$ , these procedures can be defined similarly. We first define the following procedure (denoted by `CheckZero`) to check that all values are zero in constant small communication.

- `CheckZero`  $([x_1], \dots, [x_\ell])$ : On input authenticated values  $[x_1], \dots, [x_\ell]$ ,  $P_A$  convinces  $P_B$  that  $x_i = 0$  for all  $i \in [1, \ell]$  as follows:
  1.  $P_A$  sends  $h := H(M_A[x_1], \dots, M_A[x_\ell])$  to  $P_B$ , where  $H : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa$  is a random oracle.
  2.  $P_B$  computes  $h' := H(K_B[x_1], \dots, K_B[x_\ell])$  and checks that  $h = h'$ . If the check fails,  $P_B$  aborts.

Following previous works [15, 38], we have the following lemma.

**Lemma 1.** *If  $\Delta \in \mathbb{F}_{2^\kappa}$  is sampled uniformly at random, then the probability that there exists some  $i \in [1, \ell]$  such that  $x_i \neq 0$  and  $P_B$  accepts in the `CheckZero` procedure is bounded by  $\frac{2}{2^\kappa}$ .*

The above lemma can be relaxed by allowing that  $\Delta$  is sampled uniformly from a set  $\mathcal{R} \subset \mathbb{F}_{2^\kappa}$ . In this case, the success probability for a cheating party  $P_A$  is at most  $\frac{1}{|\mathcal{R}|} + \frac{1}{2^\kappa}$ . Based on the `CheckZero` procedure, we define the following batch-opening procedure (denoted by `Open`):

**Functionality  $\mathcal{F}_{\text{bCOT}}^L$**

This functionality is parameterized by an integer  $L \geq 1$ . Running with a sender  $P_A$ , a receiver  $P_B$  and an ideal adversary, it operates as follows.

**Initialize.** Upon receiving  $(\text{init}, \text{sid}, \Delta_1, \dots, \Delta_L)$  from  $P_A$  and  $(\text{init}, \text{sid})$  from  $P_B$  where  $\Delta_i \in \mathbb{F}_{2^\kappa}$  for all  $i \in [1, L]$ , store  $(\text{sid}, \Delta_1, \dots, \Delta_L)$  and then ignore all subsequent  $(\text{init}, \text{sid})$  commands.

**Extend.** Upon receiving  $(\text{extend}, \text{sid}, \ell)$  from  $P_A$  and  $P_B$ , do the following:

- For  $i \in [1, L]$ , if  $P_A$  is honest, sample  $K_A[\mathbf{u}]_{\Delta_i} \leftarrow \mathbb{F}_{2^\kappa}^\ell$ ; otherwise, receive  $K_A[\mathbf{u}]_{\Delta_i} \in \mathbb{F}_{2^\kappa}^\ell$  from the adversary.
- If  $P_B$  is honest, sample  $\mathbf{u} \leftarrow \mathbb{F}_2^\ell$  and compute  $M_B[\mathbf{u}]_{\Delta_i} := K_A[\mathbf{u}]_{\Delta_i} + \mathbf{u} \cdot \Delta_i \in \mathbb{F}_{2^\kappa}^\ell$  for  $i \in [1, L]$ . Otherwise, receive  $\mathbf{u} \in \mathbb{F}_2^\ell$  and  $M_B[\mathbf{u}]_{\Delta_i} \in \mathbb{F}_{2^\kappa}^\ell$  for  $i \in [1, L]$  from the adversary, and recomputes  $K_A[\mathbf{u}]_{\Delta_i} := M_B[\mathbf{u}]_{\Delta_i} + \mathbf{u} \cdot \Delta_i \in \mathbb{F}_{2^\kappa}^\ell$  for  $i \in [1, L]$ .
- For  $i \in [1, L]$ , output  $(\text{sid}, K_A[\mathbf{u}]_{\Delta_i})$  to  $P_A$  and  $(\text{sid}, \mathbf{u}, M_B[\mathbf{u}]_{\Delta_i})$  to  $P_B$ .

**Fig. 1.** Functionality for block correlated oblivious transfer.

- **Open** $([x_1], \dots, [x_\ell])$ : On input authenticated values  $[x_1], \dots, [x_\ell]$  defined over field  $\mathbb{F}_{2^\kappa}$ ,  $P_A$  opens these values as follows:
  1.  $P_A$  sends  $(x_1, \dots, x_\ell)$  to  $P_B$ , and then both parties set  $[y_i] := [x_i] + x_i$  for each  $i \in [1, \ell]$ .
  2.  $P_A$  runs **CheckZero** $([y_1], \dots, [y_\ell])$  with  $P_B$ . If  $P_B$  does not abort, it outputs  $(x_1, \dots, x_\ell)$ .

### 2.3 Correlated Oblivious Transfer

Our 2PC protocol will adopt the standard functionality [10, 43] of correlated oblivious transfer (COT) to generate random authenticated bits. This functionality (denoted by  $\mathcal{F}_{\text{COT}}$ ) is shown in Fig. 1 by setting a parameter  $L = 1$ , where the extension phase can be executed multiple times for the same session identifier  $\text{sid}$ . Based on Learning Parity with Noise (LPN) [8], the recent protocols [9, 10, 13, 43] with *sublinear* communication and *linear* computation can securely realize the COT functionality in the presence of malicious adversaries. In particular, these protocols can generate a COT correlation with amortized communication cost of about  $0.1 \sim 0.4$  bits.

We also generalize the COT functionality into block COT (bCOT) [16], which allows to generate authenticated bits with the same choice bits and different global keys. Functionality  $\mathcal{F}_{\text{bCOT}}^L$  shown in Fig. 1 is the same as the standard COT functionality, except that  $L$  vectors (rather than a single vector) of authenticated bits  $[\mathbf{u}]_{B, \Delta_1}, \dots, [\mathbf{u}]_{B, \Delta_L}$  are generated. Here the vector of choice bits  $\mathbf{u}$  is required to be identical in different vectors of authenticated bits. It is easy to see that  $\mathcal{F}_{\text{COT}}$  is a special case of  $\mathcal{F}_{\text{bCOT}}^L$  with  $L = 1$ . The protocol that securely realizes functionality  $\mathcal{F}_{\text{bCOT}}^L$  is easy to be constructed by extending the LPN-based COT protocol as described above. Specifically, we set  $\Delta = (\Delta_1, \dots, \Delta_L) \in \mathbb{F}_{2^\kappa}^L \cong \mathbb{F}_{2^{\kappa L}}$  as the global key in the LPN-based COT protocol, and the resulting choice-bits are authenticated over extension field  $\mathbb{F}_{2^{\kappa L}}$ .



**Functionality  $\mathcal{F}_{\text{DVZK}}$**

This functionality runs with a prover  $\mathcal{P}$  and a verifier  $\mathcal{V}$ , and operates as follows:

- Upon receiving  $(\text{dvzk}, \text{sid}, \ell, \{\{x_i, y_i, z_i\}_{i \in [1, \ell]}\})$  from  $\mathcal{P}$  and  $\mathcal{V}$  where  $x_i, y_i, z_i \in \mathbb{F}_{2^\kappa}$  for all  $i \in [1, \ell]$ , if there exists some  $i \in [1, \ell]$  such that one of  $\{x_i, y_i, z_i\}$  is not valid, output  $(\text{sid}, \text{false})$  to  $\mathcal{V}$  and abort.
- Check that  $z_i = x_i \cdot y_i \in \mathbb{F}_{2^\kappa}$  for all  $i \in [1, \ell]$ . If the check passes, then output  $(\text{sid}, \text{true})$  to  $\mathcal{V}$ , else output  $(\text{sid}, \text{false})$  to  $\mathcal{V}$ .

**Fig. 2.** Functionality for DVZK proofs of authenticated multiplication triples.

Note that the protocol to generate block COTs still has *sublinear* communication, if  $L$  is sublinear to the number of the resulting COT correlations.

While the COT functionality outputs random authenticated bits, we can convert them into chosen authenticated bits via the following procedure (denoted by  $\text{Fix}$ ), which is also used in the recent DVZK protocol [4].

- $([x]_{\mathbb{B}, \Delta_1}, \dots, [x]_{\mathbb{B}, \Delta_L}) \leftarrow \text{Fix}(\text{sid}, \mathbf{x})$ : On input a session identifier  $\text{sid}$  of  $\mathcal{F}_{\text{bCOT}}^L$ , and a vector  $\mathbf{x} \in \mathbb{F}_2^\ell$  from  $\mathbb{P}_{\mathbb{B}}$ , two parties  $\mathbb{P}_{\mathbb{A}}$  and  $\mathbb{P}_{\mathbb{B}}$  execute the following:
  1. Both parties call  $\mathcal{F}_{\text{bCOT}}^L$  on input  $(\text{extend}, \text{sid}, \ell)$  to obtain  $([r]_{\mathbb{B}, \Delta_1}, \dots, [r]_{\mathbb{B}, \Delta_L})$  with a random vector  $\mathbf{r} \in \mathbb{F}_2^\ell$  held by  $\mathbb{P}_{\mathbb{B}}$ , where  $\mathcal{F}_{\text{bCOT}}^L$  has been initialized by  $\text{sid}$  and  $(\Delta_1, \dots, \Delta_L)$ .
  2.  $\mathbb{P}_{\mathbb{B}}$  sends  $\mathbf{d} := \mathbf{x} \oplus \mathbf{r}$  to  $\mathbb{P}_{\mathbb{A}}$ .
  3. For each  $i \in [1, L]$ , both parties set  $[x]_{\mathbb{B}, \Delta_i} := [r]_{\mathbb{B}, \Delta_i} \oplus \mathbf{d}$ .

For a field element  $x \in \mathbb{F}_{2^\kappa}$ ,  $\mathbb{P}_{\mathbb{A}}$  and  $\mathbb{P}_{\mathbb{B}}$  can run  $\mathbf{x} \leftarrow \text{F2B}(x)$ ,  $([x]_{\mathbb{B}, \Delta_1}, \dots, [x]_{\mathbb{B}, \Delta_L}) \leftarrow \text{Fix}(\text{sid}, \mathbf{x})$  and  $([x]_{\mathbb{B}, \Delta_1}, \dots, [x]_{\mathbb{B}, \Delta_L}) \leftarrow \text{B2F}([x]_{\mathbb{B}, \Delta_1}, \dots, [x]_{\mathbb{B}, \Delta_L})$  to obtain the corresponding authenticated values. Note that  $\text{B2F}$  only involves the operations multiplied by public elements  $X, \dots, X^{\kappa-1} \in \mathbb{F}_{2^\kappa}$ , and thus  $([x]_{\mathbb{B}, \Delta_1}, \dots, [x]_{\mathbb{B}, \Delta_L})$  can be computed locally by running  $\text{B2F}$ . For simplicity, we abuse the  $\text{Fix}$  notation, and use  $([x]_{\mathbb{B}, \Delta_1}, \dots, [x]_{\mathbb{B}, \Delta_L}) \leftarrow \text{Fix}(\text{sid}, x)$  to denote the conversion procedure. The  $\text{Fix}$  procedure is easy to be generalized to support that the values are defined over any field  $\mathbb{F}$  such as  $\mathbb{F} = \mathbb{F}_{2^\rho}$ . The  $\text{Fix}$  procedure is totally similar for generating authenticated bits  $[x]_{\mathbb{A}, \Delta_1}, \dots, [x]_{\mathbb{A}, \Delta_L}$  from random authenticated bits, where here  $\mathbb{P}_{\mathbb{B}}$  holds  $(\Delta_1, \dots, \Delta_L)$ . When the context is clear, we just write  $([x]_{\Delta_1}, \dots, [x]_{\Delta_L}) \leftarrow \text{Fix}(\text{sid}, \mathbf{x})$  for simplicity. We further extend  $\text{Fix}$  to additionally allow to input vectors of random authenticated bits instead of calling  $\mathcal{F}_{\text{bCOT}}^L$ , which is denoted by  $[x] \leftarrow \text{Fix}(\mathbf{x}, [\mathbf{r}])$  for the case of  $L = 1$ .

## 2.4 Designated-Verifier Zero-Knowledge Proofs

Based on IT-MACs, a family of streamable designated-verifier zero-knowledge (DVZK) proofs with fast prover time and a small memory footprint has been proposed [2–4, 17, 18, 38–41]. While these DVZK proofs can prove arbitrary circuits, we only need them to prove a simple multiplication relation. Specifically,

given a set of authenticated triples  $\{([x_i], [y_i], [z_i])\}_{i \in [1, \ell]}$  over  $\mathbb{F}_{2^\kappa}$ , these DVZK protocols can enable a prover  $\mathcal{P}$  to convince a verifier  $\mathcal{V}$  that  $z_i = x_i \cdot y_i$  for all  $i \in [1, \ell]$ . This is modeled by an ideal functionality shown in Fig. 2. In this functionality, an authenticated value  $[x]$  is input by two parties  $\mathcal{P}$  and  $\mathcal{V}$ , meaning that  $\mathcal{P}$  inputs  $(x, M)$  and  $\mathcal{V}$  inputs  $(K, \Delta)$ . We say that  $[x]$  is valid, if  $M = K + x \cdot \Delta$ . Using the recent DVZK proofs, this functionality can be *non-interactively* realized in the random-oracle model using constant small communication (e.g.,  $2\kappa$  bits in total [41]).

### 3 Technical Overview

In this section, we give an overview of our techniques. The detailed protocols and their formal proofs are described in later sections. Firstly, we recall the basic approach in the state-of-the-art solution [16].

#### 3.1 Overview of the State-of-the-Art Solution

Recently, Dittmer, Ishai, Lu and Ostrovsky [16] constructed the state-of-the-art 2PC protocol with malicious security (denoted by DILO) from simple VOLE correlations.<sup>1</sup> For one-way communication, this protocol takes  $5\rho + 1$  bits to generate a single authenticated AND triple and  $2\kappa + 3\rho$  bits per AND gate to produce one distributed garbled circuit. Their approach is outlined as follows.

In the framework of authenticated garbling [36], for each AND gate  $(i, j, k, \wedge)$ , the garbler  $P_A$  and evaluator  $P_B$  need to generate one authenticated triple  $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$  such that  $\hat{a}_k \oplus \hat{b}_k = (a_i \oplus b_i) \wedge (a_j \oplus b_j)$ . Let  $\mathbf{b} \in \mathbb{F}_2^n$  (resp.,  $\mathbf{b}_T \in \mathbb{F}_2^m$ ) be the vector of random masks  $\{b_i\}$  held by  $P_B$  on the output wires of all AND gates (resp., on all circuit-input wires associated with the  $P_B$ 's input), where  $n$  is the number of all AND gates and  $m$  is the number of all circuit-input gates. The key observation by Dittmer et al. [16] is that only evaluator  $P_B$  can compute masked wire values (i.e., the XOR of actual wire values and random masks), and thus  $\mathbf{b}$  is unnecessary to be uniformly random if the masked wire values are *not* revealed to  $P_A$ . In particular, when these masked wire values are not revealed by  $P_B$ , a malicious garbler  $P_A$  can only guess some masked wire values by performing a selective-failure attack. This means that for each masked wire value,  $P_A$  can guess correctly with probability  $1/2$ , and the protocol execution will abort for an incorrect guess. In this case,  $P_A$  can guess at most  $\rho - 1$  masked wire values, and otherwise the protocol will abort with probability at least  $1 - 1/2^\rho$ . The core idea of DILO is to compress vector  $\mathbf{b}$  by defining  $\mathbf{b} = \mathbf{M} \cdot \mathbf{b}^*$ , where  $\mathbf{M} \in \mathbb{F}_2^{n \times L}$  is a public matrix such that any  $\rho$  rows of  $\mathbf{M}$  are linearly independent,  $\mathbf{b}^* \in \mathbb{F}_2^L$  is a uniform vector and  $L = O(\rho \log(n/\rho))$ . Since IT-MACs are additively homomorphic, two parties only need to generate  $[\mathbf{b}^*]$  (instead of  $[\mathbf{b}]$ ) for a much shorter vector  $\mathbf{b}^*$ , and then compute  $[\mathbf{b}] := \mathbf{M} \cdot [\mathbf{b}^*]$ .

<sup>1</sup> VOLE is an arithmetic generalization of COT, and enables  $P_A$  to obtain  $(\Delta, K[\mathbf{u}]) \in \mathbb{F} \times \mathbb{F}^\ell$  and  $P_B$  to get  $(\mathbf{u}, M[\mathbf{u}]) \in \mathbb{F}^\ell \times \mathbb{F}^\ell$  such that  $M[\mathbf{u}] = K[\mathbf{u}] + \mathbf{u} \cdot \Delta$ , where  $\mathbb{F}$  is a large field such as  $\mathbb{F} = \mathbb{F}_{2^\rho}$ .

Dittmer et al. [16] assume that  $\mathbf{b}_{\mathcal{T}}$  is uniform and authenticated AND triples related to  $\mathbf{b}_{\mathcal{T}}$  are generated using the previous approach such as [29]. Therefore, we only show how to generate compressed authenticated AND triples, where random masks held by  $\mathsf{P}_B$  are compressed. Two parties can first generate compressed authenticated AND triple  $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$  for each AND gate with  $\Delta_A \leftarrow \mathbb{F}_{2^\rho}$ , and then convert them into that with  $\Delta'_A \leftarrow \mathbb{F}_{2^\kappa}$  using extra 2 bits of communication per AND gate, where a  $\rho$ -bit global key can guarantee that communication only depends on  $\rho$  rather than  $\kappa$  and  $\Delta'_A \in \mathbb{F}_{2^\kappa}$  is required for garbled circuits. In the following, we give an overview of Dittmer et al.'s approach on how to generate circuit-dependent compressed authenticated AND triples  $\{([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])\}$  with  $\Delta_A, \Delta_B \in \mathbb{F}_{2^\rho}$ .

1.  $\mathsf{P}_A$  and  $\mathsf{P}_B$  generates a vector of authenticated bits  $[\mathbf{b}^*]$  with a uniform  $\mathbf{b}^* \in \mathbb{F}_2^L$  by calling  $\mathcal{F}_{\text{COT}}$ . Then, both parties define  $[\mathbf{b}] := \mathbf{M} \cdot [\mathbf{b}^*]$ .
2. Both parties compute authenticated bit  $[b_{i,j}]$  for each AND gate  $(i, j, k, \wedge)$  via running the Fix procedure with input  $\{b_{i,j}\}$  where  $b_{i,j} := b_i \cdot b_j$ .
3.  $\mathsf{P}_B$  samples  $\Delta_B, \gamma \leftarrow \mathbb{F}_{2^\rho}$ . Then, both parties initializes two functionalities  $\mathcal{F}_{\text{bCOT}}^{L+2}$  and  $\mathcal{F}_{\text{bVOLE}}^{L+2}$  with the same global keys  $(b_1^* \cdot \Delta_B + \gamma, \dots, b_L^* \cdot \Delta_B + \gamma, \Delta_B + \gamma, \gamma)$ , where  $\mathcal{F}_{\text{bVOLE}}^{L+2}$  is the same as  $\mathcal{F}_{\text{bCOT}}^{L+2}$  except that the outputs are VOLE correlations over  $\mathbb{F}_{2^\rho}$  instead of COT correlations. Here  $\gamma$  is necessary to mask  $b_i^* \cdot \Delta_B$ . In particular, a consistency check in DILO lets  $\mathsf{P}_B$  send a hashing of values related to  $b_i^* \cdot \Delta_B$  to the malicious party  $\mathsf{P}_A$ , which may leak the bit  $b_i^*$  to  $\mathsf{P}_A$ . This attack would be prevented by using a uniform  $\gamma$  to mask  $b_i^* \cdot \Delta_B$ . Given  $[a]_{b_i^* \Delta_B + \gamma}$  and  $[a]_\gamma$  for any bit  $a$  held by  $\mathsf{P}_A$ , it is easy to locally compute  $[ab_i^*]_{\Delta_B}$  from the additive homomorphism of IT-MACs. Similarly, given  $[a]_{\Delta_B + \gamma}$  and  $[a]_\gamma$ , two parties can locally compute  $[a]_{\Delta_B}$ .
4.  $\mathsf{P}_A$  and  $\mathsf{P}_B$  calls  $\mathcal{F}_{\text{bCOT}}^{L+2}$  to generate the vectors of authenticated bits  $[\mathbf{a}], [\hat{\mathbf{a}}]$  as well as  $[a_i \mathbf{b}^*]_{\Delta_B}$  for each  $i \in [1, n]$ , where  $\mathbf{a} \in \mathbb{F}_2^n$  (resp.,  $\hat{\mathbf{a}} \in \mathbb{F}_2^n$ ) is used as the vector of random masks  $\{a_i\}$  (resp.,  $\{\hat{a}_k\}$ ) held by  $\mathsf{P}_A$  on the output wires of all AND gates. Then, they can locally compute  $[a_i b_j]_{\Delta_B}$  and  $[a_j b_i]_{\Delta_B}$  for each AND gate  $(i, j, k, \wedge)$  by calculating  $\mathbf{M} \cdot [a_i \mathbf{b}^*]_{\Delta_B}$ . Both parties run the Fix procedure with input  $\{a_{i,j}\}$  to obtain  $\{[a_{i,j}]\}$ , where  $a_{i,j} = a_i \wedge a_j$  for each AND gate  $(i, j, k, \wedge)$ .
5.  $\mathsf{P}_A$  and  $\mathsf{P}_B$  call  $\mathcal{F}_{\text{bVOLE}}^{L+2}$  to get a vector of authenticated values  $[\tilde{\mathbf{a}}]$  with a uniform vector  $\tilde{\mathbf{a}} \in \mathbb{F}_{2^\rho}^n$ . Both parties run the Fix procedure with input  $(\Delta_A \cdot \mathbf{a}, \Delta_A \cdot \hat{\mathbf{a}}, \{\Delta_A \cdot a_{i,j}\}, \Delta_A)$  to obtain authenticated values  $[\Delta_A \cdot \mathbf{a}], [\Delta_A \cdot \hat{\mathbf{a}}], \{[\Delta_A \cdot a_{i,j}]\}$  and  $[\Delta_A]_{\Delta_B}$ . The Fix procedure corresponds to calling  $\mathcal{F}_{\text{bVOLE}}^{L+2}$ , and also outputs  $[\Delta_A a_i \mathbf{b}^*]_{\Delta_B}$  for each  $i \in [1, n]$  and  $[\Delta_A]_{b_i^* \Delta_B}$  for each  $i \in [1, L]$  to both parties. Note that  $[\Delta_A]_{\Delta_B}$  and  $[\Delta_A]_{b_i^* \Delta_B}$  can be written as  $[\Delta_B]$  and  $[b_i^* \Delta_B]$  respectively, where we also use  $[B_i^*]$  to denote  $[b_i^* \Delta_B]$ . Furthermore,  $\mathsf{P}_A$  and  $\mathsf{P}_B$  can locally compute  $[\Delta_A a_i b_j]_{\Delta_B}$  and  $[\Delta_A a_j b_i]_{\Delta_B}$  for each AND gate  $(i, j, k, \wedge)$  by computing  $\mathbf{M} \cdot [\Delta_A a_i \mathbf{b}^*]_{\Delta_B}$  for each  $i \in [1, n]$ .
6. Parties  $\mathsf{P}_A$  and  $\mathsf{P}_B$  call  $\mathcal{F}_{\text{DVZK}}$  to prove the following relations:
  - For each AND gate  $(i, j, k, \wedge)$ , given  $([b_i], [b_j], [b_{i,j}])$ , prove  $b_{i,j} = b_i \wedge b_j$ .
  - For each AND gate  $(i, j, k, \wedge)$ , given  $([a_i], [a_j], [a_{i,j}])$ , prove  $a_{i,j} = a_i \wedge a_j$ .
  - For each  $i \in [1, L]$ , given  $([b_i^*], [\Delta_B], [B_i^*])$ , prove  $B_i^* = b_i^* \cdot \Delta_B$ .

7.  $\mathsf{P}_B$  also executes an efficient verification protocol to convince  $\mathsf{P}_A$  that the same global keys are input to different functionalities  $\mathcal{F}_{\text{bCOT}}^{L+2}$  and  $\mathcal{F}_{\text{bVOLE}}^{L+2}$ . It is unnecessary to check the consistency of  $\Delta_A \cdot \mathbf{a}, \Delta_A \cdot \hat{\mathbf{a}}, \{\Delta_A \cdot a_{i,j}\}, \Delta_A$  input to  $\text{Fix}$  w.r.t.  $\mathcal{F}_{\text{bVOLE}}^{L+2}$ . The resulting VOLE correlations on these inputs are used to compute the MAC tags of  $\mathsf{P}_B$  on its shares. If these inputs are incorrect, this only leads to these MAC tags, which will be authenticated by  $\mathsf{P}_A$ , being incorrect. This is harmless for security.
8. For each AND gate  $(i, j, k, \wedge)$ ,  $\mathsf{P}_A$  and  $\mathsf{P}_B$  locally compute  $[\tilde{b}_k]_{\Delta_B} := [a_{i,j}] + [a_i b_j] + [a_j b_i] + [\hat{a}_k]$  and  $[\tilde{B}_k]_{\Delta_B} := [\Delta_A a_{i,j}] + [\Delta_A a_i b_j] + [\Delta_A a_j b_i] + [\Delta_A \hat{a}_k] + [\tilde{a}_k]$ , where all values are authenticated under  $\Delta_B$ . Then,  $\mathsf{P}_A$  sends a pair of MAC tags  $(\mathsf{M}_A[\tilde{b}_k], \mathsf{M}_A[\tilde{B}_k])$  to  $\mathsf{P}_B$ , who computes the following over  $\mathbb{F}_{2^\rho}$

$$\tilde{b}_k := (\mathsf{K}_B[\tilde{b}_k] + \mathsf{M}_A[\tilde{b}_k]) \cdot \Delta_B^{-1} \text{ and } \tilde{B}_k := (\mathsf{K}_B[\tilde{B}_k] + \mathsf{M}_A[\tilde{B}_k]) \cdot \Delta_B^{-1}.$$

It is easy to see that  $\tilde{b}_k = a_{i,j} \oplus a_i b_j \oplus a_j b_i \oplus \hat{a}_k \in \{0, 1\}$  and  $\tilde{B}_k = (a_{i,j} + a_i b_j + a_j b_i + \hat{a}_k) \cdot \Delta_A + \tilde{a}_k \in \mathbb{F}_{2^\rho}$ , where the randomness  $\tilde{a}_k \in \mathbb{F}_{2^\rho}$  is crucial to prevent that  $\tilde{B}_k$  directly reveals  $\Delta_A$  in the case of  $\tilde{b}_k = 1$ . We observe that both parties now obtain an authenticated bit  $[\tilde{b}_k]_{\Delta_A}$  by defining its local key  $\mathsf{K}_A[\tilde{b}_k] = \tilde{a}_k$  and MAC tag  $\mathsf{M}_B[\tilde{b}_k] = \tilde{B}_k$ .

9. For each AND gate  $(i, j, k, \wedge)$ ,  $\mathsf{P}_A$  and  $\mathsf{P}_B$  locally compute an authenticated bit  $[\hat{b}_k]_{\Delta_A} := [\tilde{b}_k]_{\Delta_A} \oplus [b_{i,j}]_{\Delta_A}$ . Now, both parties obtain an authenticated triple  $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$  for each AND gate  $(i, j, k, \wedge)$ .

### 3.2 Our Solution for Generating Authenticated AND Triples

In the DILO protocol [16], the one-way communication cost of generating the authenticated triple  $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$  for each AND gate  $(i, j, k, \wedge)$  is brought about by producing an authenticated bit  $[\tilde{b}_k]$  under  $\Delta_A$  that is in turn used to locally compute  $[\hat{b}_k]$  with  $\hat{b}_k = \tilde{b}_k \oplus b_i b_j$ . DILO generates the authenticated bit  $[\tilde{b}_k] = (\mathsf{K}_A[\tilde{b}_k], \mathsf{M}_B[\tilde{b}_k], \tilde{b}_k)$  under  $\Delta_A$  by computing authenticated values on  $\tilde{b}_k$  and  $\mathsf{M}_B[\tilde{b}_k]$  under  $\Delta_B$ . Specifically, we have the following two parts:

- $\mathsf{P}_B$  computes the bit  $\tilde{b}_k$  from the authenticated bit on  $\tilde{b}_k$  under  $\Delta_B$  and corresponding MAC tag sent by  $\mathsf{P}_A$  in communication of  $\rho + 1$  bits.
- $\mathsf{P}_B$  computes the MAC tag  $\mathsf{M}_B[\tilde{b}_k]$  by generating the authenticated value on  $\mathsf{M}_B[\tilde{b}_k]$  under  $\Delta_B$  and corresponding MAC tag sent by  $\mathsf{P}_A$  in communication of  $4\rho$  bits.

We observe that the communication cost of the first part can be further reduced to only 2 bits by setting  $\text{lsb}(\Delta_B) = 1$ . In particular,  $\mathsf{P}_A$  can send the LSB  $x_k$  of the MAC tag w.r.t.  $[\tilde{b}_k]_{\Delta_B}$  to  $\mathsf{P}_B$  who can compute  $\tilde{b}_k$  by XORing  $x_k$  with the LSB of the local key w.r.t.  $[\tilde{b}_k]_{\Delta_B}$ . The authentication of  $\{\tilde{b}_k\}$  can be done in a batch by hashing the MAC tags on these bits. However, the communication cost of the second part is inherent due to the DILO approach of generating the MAC tag  $\mathsf{M}_B[\tilde{b}_k]$ . This leaves us a challenge problem: *how to generate authenticated bit  $[\tilde{b}_k]_{\Delta_A}$  without the  $\rho$ -time blow-up in communication.*

The crucial point for solving the above problem is to generate the MAC tag  $M_B[\tilde{b}_k]$  with constant communication per triple. In a straightforward way,  $P_A$  and  $P_B$  can run the `Fix` procedure to generate  $[\tilde{b}_k]_{\Delta_A}$  by taking one-bit communication after  $P_B$  has obtained  $\tilde{b}_k$ . However,  $P_A$  has no way to check the correctness of  $\tilde{b}_k$  implied in  $[\tilde{b}_k]_{\Delta_A}$ , where  $[\tilde{b}_k]_{\Delta_B}$  generated by both parties only allow  $P_B$  to check the correctness of  $\tilde{b}_k$ . We introduce the notion of dual-key authentication to allow both parties to check the correctness of  $\tilde{b}_k$ , where the bit  $\tilde{b}_k$  is authenticated under global key  $\Delta_A \cdot \Delta_B$  and thus no party can change the bit  $\tilde{b}_k$  without being detected. We present an efficient approach to generate the dual-key authenticated bit  $\langle \tilde{b}_k \rangle$  with communication of only one bit. By checking the consistency of all values input to the block-COT functionality, we can guarantee the correctness of  $\langle \tilde{b}_k \rangle$ , i.e.,  $\tilde{b}_k$  is a valid bit authenticated by both parties. When setting  $\text{lsb}(\Delta_A \cdot \Delta_B) = 1$ ,  $P_B$  can obtain the bit  $\tilde{b}_k$  by letting  $P_A$  send one-bit message to  $P_B$  (see below for details). By using `Fix`,  $P_A$  and  $P_B$  can generate  $[\tilde{b}_k]$  under  $\Delta_A$ . Now,  $P_B$  can check the correctness of  $\tilde{b}_k$  obtained, and  $P_A$  can verify the correctness of  $\tilde{b}_k$  implied in  $[\tilde{b}_k]$ , by using the correctness of  $\langle \tilde{b}_k \rangle$ . Particularly, we propose a batch-check technique that enables both parties to check the correctness of  $\langle \tilde{b}_k \rangle$  in all triples with essentially no communication. In addition, we present two new checking protocols to verify the correctness of global keys and the consistency of values across different functionalities (see below for an overview). Overall, our techniques allow to achieve one-way communication of only 2 bits per triple, and are described below.

*Dual-key Authentication.* We propose the notion of dual-key authentication, meaning that a bit is authenticated by two global keys  $\Delta_A, \Delta_B \in \mathbb{F}_{2^\kappa}$  held by  $P_A$  and  $P_B$  respectively. In particular, a dual-key authenticated bit  $\langle x \rangle = (D_A[x], D_B[x], x)$  lets  $P_A$  hold  $D_A[x]$  and  $P_B$  hold  $D_B[x]$  such that  $D_A[x] + D_B[x] = x \cdot \Delta_A \cdot \Delta_B \in \mathbb{F}_{2^\kappa}$ , where  $x \in \{0, 1\}$  can be known by either  $P_A$  or  $P_B$ , or unknown for both parties. From the definition, we have that dual-key authenticated bits are also *additively homomorphic*, which enables us to use the random-linear-combination approach to perform consistency checks associated with such bits. We are also able to generalize dual-key authenticated bits to dual-key authenticated values in which  $x$  is defined over any field  $\mathbb{F}$  and  $D_A[x], D_B[x], \Delta_A, \Delta_B$  are defined over an extension field  $\mathbb{K}$  with  $\mathbb{F} \subseteq \mathbb{K}$ . This generalization may be useful for the design of subsequent protocols. A useful property is that  $\langle x \rangle$  can be *locally* converted into  $[x\Delta_A]_{\Delta_B}$  or  $[x\Delta_B]_{\Delta_A}$  and vice versa.

We consider that the bit  $x$  is shared as  $(a, b)$  with  $x = a \wedge b$ , where  $P_A$  holds  $a \in \{0, 1\}$  and  $P_B$  holds  $b \in \{0, 1\}$ . Without loss of generality, we focus on the case that  $a$  is a secret bit. The bit  $b$  can be either a secret bit or a public bit 1, where the former means that no party knows  $x$  and the latter means that only  $P_A$  knows  $x$ . The DILO protocol [16] implicitly generates a dual-key authenticated bit by running `Fix`( $a\Delta_A$ ) w.r.t. global keys  $b\Delta_B + \gamma, \gamma$  to obtain  $[a\Delta_A]_{b\Delta_B} = \langle ab \rangle = \langle x \rangle$ , which incurs  $\rho$ -time blow-up in communication (even if  $a$  allows to be a random bit). Our approach can reduce the communication cost to at most one bit. In particular, we first let  $P_A$  and  $P_B$  generate a dual-key authenticated bit  $\langle b \rangle = (\alpha, \beta)$  with  $\alpha + \beta = b \cdot \Delta_A \cdot \Delta_B \in \mathbb{F}_{2^\kappa}$ , where  $P_A$  gets  $\alpha$

and  $P_B$  obtains  $\beta$ . Then, both parties initialize functionality  $\mathcal{F}_{\text{bCOT}}$  with a global key  $\beta$ . If  $a \in \{0, 1\}$  allows to be random, both parties call  $\mathcal{F}_{\text{bCOT}}$  to generate  $[a]_\beta$  without communication. Otherwise, both parties run  $\text{Fix}$  with input  $a$  to generate  $[a]_\beta$  in communication of one bit. Given  $[a]_\beta = (\mathbf{K}_B[a]_\beta, \mathbf{M}_A[a]_\beta, a)$ ,  $P_A$  and  $P_B$  can *locally* compute a dual-key authenticated bit  $\langle a \rangle$  by letting  $P_A$  compute  $D_A[x] := \mathbf{M}_A[a]_\beta + a \cdot \alpha \in \mathbb{F}_{2^\kappa}$  and  $P_B$  set  $D_B[x] := \mathbf{K}_B[a]_\beta \in \mathbb{F}_{2^\kappa}$ . We have that  $D_A[x] + D_B[x] = (\mathbf{M}_A[a]_\beta + \mathbf{K}_B[a]_\beta) + a \cdot \alpha = a \cdot (\alpha + \beta) = a \cdot b \cdot \Delta_A \cdot \Delta_B \in \mathbb{F}_{2^\kappa}$ . To guarantee correctness of  $\langle x \rangle$ , we need to check the consistency of  $\beta$  input to  $\mathcal{F}_{\text{bCOT}}$  and  $a$  input to  $\text{Fix}$ , which will be shown below.

*Sampling Global Keys with Correctness Checking.* As described above, we need to generate two global keys  $\Delta_A$  and  $\Delta_B$  such that  $\text{lsb}(\Delta_A \cdot \Delta_B) = 1$ , which allows one party to get the bit  $x = \text{lsb}(D_A[x]) \oplus \text{lsb}(D_B[x])$  from a dual-key authenticated bit  $\langle x \rangle$ . To do this, we let  $P_A$  sample  $\Delta_A \leftarrow \{0, 1\}^\kappa$  such that  $\text{lsb}(\Delta_A) = 1$ . Then, we let  $P_B$  sample  $\Delta_B \leftarrow \{0, 1\}^\kappa$ , and make  $P_A$  and  $P_B$  run the  $\text{Fix}$  procedure w.r.t.  $\Delta_A$  with input  $\Delta_B$  to generate  $[\Delta_B]_{\Delta_A}$  (i.e.,  $\langle 1 \rangle$ ), where  $\alpha_0 \oplus \beta_0 = \Delta_A \Delta_B$ .  $P_A$  and  $P_B$  can exchange  $\text{lsb}(\alpha_0)$  and  $\text{lsb}(\beta_0)$  to decide whether  $\text{lsb}(\alpha_0) \oplus \text{lsb}(\beta_0) = 0$ . If yes, then  $\text{lsb}(\Delta_A \Delta_B) = \text{lsb}(\alpha_0) \oplus \text{lsb}(\beta_0) = 0$ . In this case, we let  $P_B$  update  $\Delta_B$  as  $\Delta_B \oplus 1$ , which makes  $\Delta_A \Delta_B$  be updated as  $\Delta_A \Delta_B \oplus \Delta_A$ , where  $\text{lsb}(\Delta_A \Delta_B \oplus \Delta_A) = \text{lsb}(\Delta_A \Delta_B) \oplus \text{lsb}(\Delta_A) = 1$ . Since  $\Delta_B$  is changed as  $\Delta_B \oplus 1$ ,  $\alpha_0$  needs to be updated as  $\alpha_0 \oplus \Delta_A$  in order to keep correct correlation.

While we adopt the KRRW authenticated garbling [29] in dual executions, some bit of global keys  $\Delta_A, \Delta_B \in \{0, 1\}^\kappa$  is required to be fixed as 1. We often choose to define  $\text{lsb}(\Delta_A) = 1$  and  $\text{lsb}(\Delta_B) = 1$ . While  $\text{lsb}(\Delta_A) = 1$  has been satisfied,  $\text{lsb}(\Delta_B) = 1$  does not always hold, as  $P_B$  may flip  $\Delta_B$  depending on if  $\text{lsb}(\alpha_0) \oplus \text{lsb}(\beta_0) = 0$ . Thus, we let  $P_B$  set  $\text{msb}(\Delta_B) = 1$  for ease of remembering. More importantly,  $\text{msb}(\Delta_B) = 1$  has no impact on setting  $\text{lsb}(\Delta_A \Delta_B) = 1$ .

To achieve active security, we need to guarantee that  $\Delta_A \cdot \Delta_B \neq 0$  in the case that either  $P_A$  or  $P_B$  is corrupted. This can be assured by checking  $\Delta_A \neq 0$  and  $\Delta_B \neq 0$ . We choose to check  $\text{lsb}(\Delta_A) = 1$  and  $\text{msb}(\Delta_B) = 1$  to realize the checking of  $\Delta_A \neq 0$  and  $\Delta_B \neq 0$ . To enable  $P_B$  to check  $\text{lsb}(\Delta_A) = 1$ , both parties can generate random authenticated bits  $[r_1]_B, \dots, [r_\rho]_B$ , and then  $P_A$  sends  $\text{lsb}(\mathbf{K}_A[r_i])$  for  $i \in [1, \rho]$  to  $P_B$  who checks that  $\text{lsb}(\mathbf{K}_A[r_i]) \oplus \text{lsb}(\mathbf{M}_B[r_i]) = r_i$  for all  $i \in [1, \rho]$ . A malicious  $P_A$  can cheat successfully if and only if it guesses correctly all random bits  $r_1, \dots, r_\rho$ , which happens with probability  $1/2^\rho$ . The correctness check of  $\text{msb}(\Delta_B) = 1$  can be done in a totally similar way. Furthermore, we need also to check  $\text{lsb}(\Delta_A \Delta_B) = 1$ , and otherwise a selective failure attack may be performed on secret bit  $b_k$ . We first let  $P_B$  check  $\text{lsb}(\Delta_A \Delta_B) = 1$  by interacting with  $P_A$ . We make  $P_A$  and  $P_B$  generate random dual-key authenticated bits  $\langle s_1 \rangle, \dots, \langle s_\rho \rangle$ . Then, the check of  $\text{lsb}(\Delta_A \Delta_B) = 1$  can be done similarly, by letting  $P_A$  send  $\text{lsb}(D_A[s_i])$  to  $P_B$  who checks that  $\text{lsb}(D_A[s_i]) \oplus \text{lsb}(D_B[s_i]) = s_i$  for all  $i \in [1, \rho]$ . To produce  $\langle s_1 \rangle, \dots, \langle s_\rho \rangle$ ,  $P_A$  and  $P_B$  can call  $\mathcal{F}_{\text{COT}}$  to generate random authenticated bits  $[s_1]_{\Delta_A}, \dots, [s_\rho]_{\Delta_A}$ , and then run the  $\text{Fix}$  procedure w.r.t.  $\Delta_A$  on input  $(s_1 \Delta_B, \dots, s_\rho \Delta_B)$  to generate  $[s_1 \Delta_B]_{\Delta_A}, \dots, [s_\rho \Delta_B]_{\Delta_A}$  that are equivalent to  $\langle s_1 \rangle, \dots, \langle s_\rho \rangle$ . Then, the correctness of the input  $(s_1 \Delta_B, \dots, s_\rho \Delta_B)$  needs to be verified by  $P_A$  via letting  $P_B$  prove that  $([s_i]_{\Delta_A}, [\Delta_B]_{\Delta_A}, [s_i \Delta_B]_{\Delta_A})$  for all  $i \in [1, \rho]$

satisfy the multiplication relationship using  $\mathcal{F}_{\text{DVZK}}$ . Due to the dual execution,  $\text{P}_A$  needs also to symmetrically check  $\text{lsb}(\Delta_A \Delta_B) = 1$  by interacting with  $\text{P}_B$ .

*Generating Compressed Authenticated AND Triples.* As described above, for generating a compressed authenticated AND triple  $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$ , the crucial step is to generate a dual-key authenticated bit  $\langle \tilde{b}_k \rangle$  with  $\tilde{b}_k = \hat{b}_k \oplus b_i b_j$ . From the definition of  $\tilde{b}_k$ , we know that  $\langle \tilde{b}_k \rangle = \langle a_{i,j} \rangle \oplus \langle a_i b_j \rangle \oplus \langle a_j b_i \rangle \oplus \langle \hat{a}_k \rangle$ . We use the above approach to generate the dual-key authenticated bits  $\langle a_{i,j} \rangle$ ,  $\langle \hat{a}_k \rangle$  and  $\langle a_i \mathbf{b}^* \rangle$  for  $i \in [1, n]$  that can be locally converted to  $\langle a_i b_j \rangle$ ,  $\langle a_j b_i \rangle$  by multiplying a public matrix  $\mathbf{M}$ . Then, we combine all the dual-key authenticated bits to obtain  $\langle \tilde{b}_k \rangle$ . From  $\text{lsb}(\Delta_A \Delta_B) = 1$ , we can let  $\text{P}_A$  send  $\text{lsb}(\text{D}_A[\tilde{b}_k])$  to  $\text{P}_B$  who is able to recover  $\tilde{b}_k = \text{lsb}(\text{D}_A[\tilde{b}_k]) \oplus \text{lsb}(\text{D}_B[\tilde{b}_k])$ . By running the Fix procedure with input  $\tilde{b}_k$ , both parties can generate  $[\tilde{b}_k]$ , which can be in turn locally converted into  $[\hat{b}_k]$ . More details are shown as follows.

1. As in the DILO protocol [16], we let  $\text{P}_A$  and  $\text{P}_B$  obtain  $\langle \mathbf{b}^* \rangle$  and  $\{\langle [b_{i,j}] \rangle\}$  by calling  $\mathcal{F}_{\text{COT}}$  and running Fix with input  $b_{i,j} = b_i b_j$ . Then, both parties compute  $[b] := \mathbf{M} \cdot \langle \mathbf{b}^* \rangle$  to obtain  $[b_i], [b_j]$  for each AND gate  $(i, j, k, \wedge)$ .
2.  $\text{P}_A$  and  $\text{P}_B$  have produced  $\langle 1 \rangle = (\alpha_0, \beta_0)$  such that  $\alpha_0 + \beta_0 = \Delta_A \cdot \Delta_B \in \mathbb{F}_{2^\kappa}$ . For each  $i \in [1, L]$ , both parties can further generate a dual-key authenticated bit  $\langle b_i^* \rangle = (\alpha_i, \beta_i)$  with  $\alpha_i + \beta_i = b_i^* \cdot \Delta_A \cdot \Delta_B \in \mathbb{F}_{2^\kappa}$  by running Fix w.r.t.  $\Delta_A$  with input  $B_i^* = b_i^* \Delta_B$ . The communication to generate  $\langle b_1^* \rangle, \dots, \langle b_L^* \rangle$  is  $L\kappa$  bits and logarithmic to the number  $n$  of AND gates due to  $L = O(\rho \log(n/\rho))$ .
3.  $\text{P}_B$  and  $\text{P}_A$  initialize  $\mathcal{F}_{\text{bCOT}}^{L+1}$  with global keys  $\beta_1, \dots, \beta_L, \Delta_B$ , and then call  $\mathcal{F}_{\text{bCOT}}^{L+1}$  to generate  $[a]_{\beta_1}, \dots, [a]_{\beta_L}$  and  $[a]_{\Delta_B}$ . For each tuple  $([a_i]_{\beta_1}, \dots, [a_i]_{\beta_L})$ , we can convert it to  $\langle a_i \mathbf{b}^* \rangle$ . By multiplying the public matrix  $\mathbf{M}$ , both parties can obtain  $\langle a_i b_j \rangle$  and  $\langle a_j b_i \rangle$  for each AND gate  $(i, j, k, \wedge)$ . From  $[a]_{\Delta_B}$ , both parties directly obtain  $[a_i], [a_j]$  for each AND gate  $(i, j, k, \wedge)$ .
4.  $\text{P}_B$  and  $\text{P}_A$  initialize  $\mathcal{F}_{\text{bCOT}}^2$  with global keys  $\beta_0, \Delta_B$ , and then call  $\mathcal{F}_{\text{bCOT}}^2$  to generate  $[\hat{a}]_{\beta_0}$  and  $[\hat{a}]_{\Delta_B}$ . Both parties further run the Fix procedure with input  $a_{i,j} = a_i \wedge a_j$  to generate  $[a_{i,j}]_{\beta_0}$  and  $[a_{i,j}]_{\Delta_B}$ , where  $[a_{i,j}]_{\Delta_B}$  will be used to prove validity of  $a_{i,j}$ . The parties can convert  $[\hat{a}]_{\beta_0}$  and  $\{[a_{i,j}]_{\beta_0}\}$  into  $\langle \hat{a}_k \rangle$  and  $\langle a_{i,j} \rangle$  for each AND gate  $(i, j, k, \wedge)$ .
5. Both parties can *locally* compute  $\langle \tilde{b}_k \rangle := \langle a_{i,j} \rangle \oplus \langle a_i b_j \rangle \oplus \langle a_j b_i \rangle \oplus \langle \hat{a}_k \rangle$ . Then,  $\text{P}_A$  can send  $\text{lsb}(\text{D}_A[\tilde{b}_k])$  to  $\text{P}_B$ , who computes  $\tilde{b}_k := \text{lsb}(\text{D}_A[\tilde{b}_k]) \oplus \text{lsb}(\text{D}_B[\tilde{b}_k])$  due to  $\text{lsb}(\Delta_A \Delta_B) = 1$ . Both parties run Fix on input  $\tilde{b}_k$  to generate  $[\tilde{b}_k]$ .
6.  $\text{P}_A$  and  $\text{P}_B$  can *locally* compute  $[\hat{b}_k] := [\tilde{b}_k] \oplus [b_{i,j}]$ . Now, the parties hold  $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$  for each AND gate  $(i, j, k, \wedge)$ .

*Consistency Check.* We have shown how to generate compressed authenticated AND triples. Below, we show how to verify their correctness. We only need to guarantee the consistency of all Fix inputs, all global keys input to the bCOT functionality and all bits sent by  $\text{P}_A$  to  $\text{P}_B$ . When all messages and inputs are consistent, no malicious party can break the correctness of all triples. Specifically, we present the following checks to guarantee the consistency.

1. Check the correctness of the following authenticated AND triples:
  - $([b_i], [b_j], [b_{i,j}])$  s.t.  $b_{i,j} = b_i \wedge b_j$  for each AND gate  $(i, j, k, \wedge)$ .
  - $([a_i], [a_j], [a_{i,j}])$  s.t.  $a_{i,j} = a_i \wedge a_j$  for each AND gate  $(i, j, k, \wedge)$ .
  - $([b_i^*], [\Delta_B], [B_i^*])$  s.t.  $B_i^* = b_i^* \cdot \Delta_B$  for each  $i \in [1, L]$ .
2. The keys  $\beta_0, \beta_1, \dots, \beta_L$  input to functionality  $\mathcal{F}_{\text{bCOT}}$  are consistent to the values defined in  $\langle 1 \rangle, \langle b_1^* \rangle, \dots, \langle b_L^* \rangle$ .
3.  $P_A$  needs to check that two global keys  $\Delta_B^{(1)}$  and  $\Delta_B^{(2)}$  respectively input to functionalities  $\mathcal{F}_{\text{bCOT}}^{L+1}$  and  $\mathcal{F}_{\text{bCOT}}^2$  are consistent with  $\Delta_B$  defined in  $\langle 1 \rangle$ .
4.  $P_A$  checks that the bit  $\tilde{b}_k$  defined in  $[b_k]$  is consistent to that defined in  $\langle \tilde{b}_k \rangle$ , and  $P_B$  checks that  $\tilde{b}_k$  computed by itself is consistent to that defined in  $\langle \tilde{b}_k \rangle$ .

The first two checks guarantee the correctness of  $\langle \tilde{b}_k \rangle$  and  $[b_{i,j}]$ , the third check verifies the consistency of the global keys in  $[a_i], [a_j], [\hat{a}_k]$ , and the final check assure the consistency of bits authenticated between  $\langle \tilde{b}_k \rangle$  and  $[b_k]$ . Check 1 can be directly realized by calling functionality  $\mathcal{F}_{\text{DVZK}}$ .

For Check 2, for each  $i \in [0, L]$ , we let  $P_A$  and  $P_B$  run the Fix procedure w.r.t.  $\beta_i$  on input  $\Delta'_A$  to generate  $[\Delta'_A]_{\beta_i}$ , which can be locally converted into  $[\beta_i]_{\Delta'_A}$ , where  $\Delta'_A \in \mathbb{F}_{2^\kappa}$  is sampled uniformly at random by  $P_A$ .<sup>2</sup> For  $i \in [0, L]$ , we present a new protocol to verify the consistency of  $\beta_i$  in the following equations:

$$\begin{aligned} \alpha_i + \beta_i &= b_i^* \cdot \Delta_A \cdot \Delta_B, \\ K'_A[\beta_i] + M'_A[\beta_i] &= \beta_i \cdot \Delta'_A, \end{aligned}$$

where  $b_0^*$  is defined as 1. We first multiply two sides of the first equation by  $\Delta_A^{-1}$ , and obtain  $\alpha_i \cdot \Delta_A^{-1} + \beta_i \cdot \Delta_A^{-1} = b_i^* \cdot \Delta_B$ . We rewrite the resulting equation as  $K_A[\beta_i] + M_B[\beta_i] = \beta_i \cdot \Delta_A^{-1}$  where  $K_A[\beta_i] = \alpha_i \cdot \Delta_A^{-1}$  and  $M_B[\beta_i] = b_i^* \cdot \Delta_B$ . Below, we can adapt the known techniques [16, 18] to check the consistency of  $\beta_i$  authenticated under different global keys (i.e.,  $[\beta_i]_{\Delta_A^{-1}}$  and  $[\beta_i]_{\Delta'_A}$ ) in a batch (see Sect. 4.3 for details).

For Check 3, we make  $P_A$  and  $P_B$  run the Fix procedure w.r.t.  $\Delta_B^{(1)}$  (resp.,  $\Delta_B^{(2)}$ ) on input  $\Delta'_A$  to obtain  $[\Delta_B^{(1)}]_{\Delta'_A}$  (resp.,  $[\Delta_B^{(2)}]_{\Delta'_A}$ ). Authenticated values  $[\Delta_B^{(1)}]_{\Delta'_A}$  and  $[\Delta_B^{(2)}]_{\Delta'_A}$  are equivalent to  $\langle 1_B^{(1)} \rangle$  and  $\langle 1_B^{(2)} \rangle$  where  $\Delta_B^{(1)} \Delta'_A$  and  $\Delta_B^{(2)} \Delta'_A$  are used as the global keys in dual-key authentication. Both parties can invoke a relaxed equality-check functionality  $\mathcal{F}_{\text{EQ}}$  (shown in the full version [14]) to check  $1_B^{(1)} - 1_B^{(2)} = 0$ . Using the checking technique by Dittmer et al. [16], we can also check the consistency of the values authenticated between  $[\Delta_B^{(1)}]_{\Delta'_A}$  and  $[\Delta_B]_{\Delta_A}$  generated during the sampling phase.

For Check 4, we use a random-linear-combination approach to perform the check in a batch. Specifically, we can let  $P_A$  and  $P_B$  call  $\mathcal{F}_{\text{COT}}$  to generate  $[r]_B$  and then obtain  $[r]_B \leftarrow \text{B2F}([r]_B)$ , where  $r \in \mathbb{F}_{2^\kappa}$  is uniform. Then, both parties run Fix w.r.t.  $\Delta_A$  on input  $r \Delta_B$  to generate  $[r \Delta_B]_{\Delta_A}$  (i.e.,  $\langle r \rangle$ ). We can let the parties call a standard coin-tossing functionality  $\mathcal{F}_{\text{Rand}}$  to sample a random element

<sup>2</sup> An independent global key  $\Delta'_A$  is necessary to perform the consistency check, and otherwise a malicious  $P_B$  will always pass the check if  $\Delta_A$  is reused.



$\chi \in \mathbb{F}_{2^\kappa}$ . Then, both parties can locally compute  $\langle y \rangle := \sum \chi^k \cdot \langle \tilde{b}_k \rangle + \langle r \rangle$  and  $[y]_{\mathbb{B}} := \sum \chi^k \cdot [\tilde{b}_k]_{\mathbb{B}} + [r]_{\mathbb{B}}$ . Then,  $P_{\mathbb{B}}$  can open  $[y]_{\mathbb{B}}$  that allows  $P_{\mathbb{A}}$  to get  $y$  in an authenticated way. Finally, both parties can use  $\mathcal{F}_{\text{EQ}}$  to verify that the opening of  $\langle y \rangle - y \cdot \langle 1 \rangle$  is 0. Since  $\chi$  is sampled uniformly at random after all authenticated values are determined, the consistency check will detect malicious behaviors except with probability at most  $n/2^\kappa$ .

### 3.3 Our Solution for Dual Execution Without Leakage

While the evaluator’s random masks are compressed, the state-of-the-art construction of authenticated garbling based on half-gates by Katz et al. [29] is no longer applied. The circuit authentication approach in [29] requires the evaluator to reveal all masked wire values, which is prohibitive for the compression technique. Therefore, based on the technique [36], Dittmer et al. [16] designed a new construction of authenticated garbling without revealing masked wire values. However, this construction incurs extra communication overhead of  $3\rho - 1$  bits per AND gate, compared to the half-gates-based construction [29].

In duplex networks, communication cost is often measured by one-way communication. This allows us to adopt the idea of dual execution [33] to perform the authentication of circuit evaluation. In the original dual execution [33], the semi-honest Yao-2PC protocol [44] is executed two times with the same inputs in parallel by swapping the roles of parties for the second execution, and then the correctness of the output is verified by checking that the two executions have the same output bits. However, an inherent problem of the above method is that selective failure attacks are allowed to leak one-bit information of the input by the honest party, even though there exists a protocol to check the consistency of inputs in two executions. For example, suppose that  $P_{\mathbb{A}}$  is honest and  $P_{\mathbb{B}}$  is malicious. When  $P_{\mathbb{A}}$  is a garbler and  $P_{\mathbb{B}}$  is an evaluator, both parties compute an output  $f(x, y)$  where  $x$  is the  $P_{\mathbb{A}}$ ’s input and  $y$  is the  $P_{\mathbb{B}}$ ’s input. After swapping the roles, they compute another output  $g(x, y)$  with  $g \neq f$ , as garbler  $P_{\mathbb{B}}$  is malicious. If the output-equality check passes, then  $g(x, y) = f(x, y)$ , else  $g(x, y) \neq f(x, y)$ . In both cases, this leaks one-bit information on the input  $x$ .

In the authenticated garbling framework, we propose a new technique to circumvent the problem and eliminate the one-bit leakage. Together with our technique to generate compressed authenticated AND triples, we can achieve the cost of one-way communication that is almost the same as the semi-honest half-gates protocol [45]. Specifically, we let  $P_{\mathbb{A}}$  and  $P_{\mathbb{B}}$  execute the protocol, which combines the sub-protocol of generating authenticated AND triples as described above with the construction of distributed garbling [29], for two times with same inputs in the dual-execution way. For each wire  $w$  in the circuit, we need to check that the actual values  $z_w$  and  $z'_w$  in two executions are identical. We perform the checking by verifying  $z_w \cdot (\Delta_{\mathbb{A}} \oplus \Delta_{\mathbb{B}}) = z'_w \cdot (\Delta_{\mathbb{A}} \oplus \Delta_{\mathbb{B}})$ . Since  $\Delta_{\mathbb{A}} \oplus \Delta_{\mathbb{B}}$  is unknown for the adversary, the probability that  $z_w \neq z'_w$  but the check passes is negligible. Our approach allows two parties to check the correctness of all wire values in the circuit, and thus prevents selective failure attacks.

In more detail, for each wire  $w$ , let  $\Lambda_w$  and  $(a_w, b_w)$  be the masked value and wire masks in the first execution and  $(\Lambda'_w, a'_w, b'_w)$  be the values in the second execution. Thus,  $\mathsf{P}_A$  and  $\mathsf{P}_B$  need to check that  $\Lambda_w \oplus a_w \oplus b_w = \Lambda'_w \oplus a'_w \oplus b'_w$  for each wire  $w$ , where the output wires of XOR gates are unnecessary to be checked as they are locally computed. Below, our task is to check that  $(\Lambda_w \oplus a_w \oplus b_w) \cdot (\Delta_A \oplus \Delta_B) = (\Lambda'_w \oplus a'_w \oplus b'_w) \cdot (\Delta_A \oplus \Delta_B)$  holds for each wire  $w$ . By two protocol executions, both parties hold  $([a_w], [b_w], [a'_w], [b'_w])$  for each wire  $w$ . When  $\mathsf{P}_A$  is a garbler and  $\mathsf{P}_B$  is an evaluator,  $\mathsf{P}_A$  holds a garbled label  $L_{w,0}$  and  $\mathsf{P}_B$  holds  $(\Lambda_w, L_{w,\Lambda_w})$ . Since  $L_{w,\Lambda_w} = L_{w,0} \oplus \Lambda_w \Delta_A$  has the form of IT-MACs, we can view  $(L_{w,0}, L_{w,\Lambda_w}, \Lambda_w)$  as an authenticated bit  $[A_w]_{\mathsf{B}}$ , where  $L_{w,0}$  is considered as the local key and  $L_{w,\Lambda_w}$  plays the role of MAC tag. Similarly, when  $\mathsf{P}_A$  is an evaluator and  $\mathsf{P}_B$  is a garbler, two parties hold an authenticated bit  $[A'_w]_{\mathsf{A}}$ . Following the known observation (e.g., [29]), for any authenticated bit  $[y]_{\mathsf{B}}$ ,  $\mathsf{P}_A$  and  $\mathsf{P}_B$  have an additive sharing of  $y \cdot \Delta_A = K_{\mathsf{A}}[y] \oplus M_{\mathsf{B}}[y]$ . Therefore, for all cross terms, both parties can obtain their additive shares, and then can compute two values that are checked to be identical. In particular, both parties can compute the additive shares of all cross terms:  $Z_{w,1}^A \oplus Z_{w,1}^B = \Lambda_w \Delta_A$ ,  $Z_{w,2}^A \oplus Z_{w,2}^B = \Lambda'_w \Delta_B$ ,  $Z_{w,3}^A \oplus Z_{w,3}^B = a_w \Delta_B$ ,  $Z_{w,4}^A \oplus Z_{w,4}^B = a'_w \Delta_B$ ,  $Z_{w,5}^A \oplus Z_{w,5}^B = b_w \Delta_A$ ,  $Z_{w,6}^A \oplus Z_{w,6}^B = b'_w \Delta_A$ . Then, for each wire  $w$ ,  $\mathsf{P}_A$  and  $\mathsf{P}_B$  can respectively compute

$$\begin{aligned} V_w^A &= (\oplus_{i \in [1,6]} Z_{w,i}^A) \oplus a_w \Delta_A \oplus \Lambda'_w \Delta_A \oplus a'_w \Delta_A \\ V_w^B &= (\oplus_{i \in [1,6]} Z_{w,i}^B) \oplus b_w \Delta_B \oplus \Lambda_w \Delta_B \oplus b'_w \Delta_B, \end{aligned}$$

such that  $V_w^A = V_w^B$ . Without loss of generality, we assume that only  $\mathsf{P}_B$  obtains the output, and thus only  $\mathsf{P}_B$  needs to check the correctness of all masked values. In this case, we make  $\mathsf{P}_A$  send the hash value of all  $V_w^A$  to  $\mathsf{P}_B$ , who can check its correctness with  $V_w^B$  for each wire  $w$ .

*Optimizations for Processing Inputs.* Dittmer et al. [16] consider that the wire masks (i.e.,  $\mathbf{b}_{\mathcal{I}}$ ) on all wires in  $\mathcal{I}_{\mathsf{B}}$  held by evaluator  $\mathsf{P}_B$  is uniformly random and authenticated AND triples associated with  $\mathbf{b}_{\mathcal{I}}$  are generated using the previous approach (e.g., [29]). This will require an independent preprocessing protocol, and also brings more preprocessing communication cost. We solve the problem by specially processing the input of evaluator  $\mathsf{P}_B$ . In particular, instead of making  $\mathsf{P}_B$  send masked value  $\Lambda_w := y_w \oplus b_w$  for each  $w \in \mathcal{I}_{\mathsf{B}}$  to  $\mathsf{P}_A$  where  $y_w$  is the input bit, we use an OT protocol to transmit  $L_{w,\Lambda_w}$  to  $\mathsf{P}_B$ . This allows to keep masked wire values  $\Lambda_w := y_w \oplus b_w$  for all  $w \in \mathcal{I}_{\mathsf{B}}$  secret. In this case, we can compress the wire masks using the technique as described in Sect. 3.2 and adopt the same preprocessing protocol to handle  $\mathbf{b}_{\mathcal{I}}$ . Since  $L$  is logarithm to the length  $n$  of vector  $\mathbf{b}$  (now  $n = |\mathcal{W}| + |\mathcal{I}_{\mathsf{B}}|$ ), this optimization essentially incurs no more overhead for the preprocessing phase. Furthermore, our preprocessing protocol to generate authenticated AND triples has already invoked functionality  $\mathcal{F}_{\text{COT}}$ . Therefore, we can let two parties call  $\mathcal{F}_{\text{COT}}$  to generate random COT correlations in the preprocessing phase, and then transform them to OT correlations in the standard way. This essentially brings no more communication for the preprocessing phase, due to the sublinear communication of the recent protocols instantiating  $\mathcal{F}_{\text{COT}}$ .

Our optimization does not increase the rounds of online phase. As a trade-off, this optimization increases the online communication cost by  $|\mathcal{I}_B| \cdot \kappa$  bits.

In the second protocol execution (i.e.,  $P_A$  as an evaluator and  $P_B$  as a garbler), we make a further optimization to directly guarantee that the masked values on all circuit-input wires are XOR of actual values and wire masks. In this case, it is unnecessary to check the correctness of masked values on all circuit-input wires between two protocol executions. The key idea is to utilize the authenticated bits and messages on circuit-input wires generated/sent during the first protocol execution along with the authenticated bits produced in the second protocol execution to generate the masked values on the wires in  $\mathcal{I}_A \cup \mathcal{I}_B$ . Due to the security of IT-MACs, we can guarantee the correctness of these masked values in the second execution. We postpone the details of this optimization to Sect. 5.

## 4 Preprocessing with Compressed Wire Masks

In this section we introduce the compressed preprocessing functionality  $\mathcal{F}_{\text{cpre}}$  (shown in Fig. 3) for two party computation as well as an efficient protocol  $\Pi_{\text{cpre}}$  (shown in Fig. 5 and Fig. 6) to realize it. In a modular fashion we first introduce the sub-components which are called in the main preprocessing protocol. The security of the protocol is also argued similarly: we first prove in separate lemmas the respective security properties of sub-components and then utilize these lemmas to prove the main theorem.

### 4.1 Dual-Key Authentication

In this subsection we define the format of dual-key authentication and list some of its properties that we utilize in the upper level preprocessing protocol.

**Definition 1.** *We use the notation  $\langle x \rangle := (D_A[x], D_B[x], x)$  to denote the dual-key authenticated value  $x$ , where  $P_A, P_B$  holds  $D_A[x], D_B[x]$  subject to  $D_A[x] + D_B[x] = x\Delta_A\Delta_B$  and  $\Delta_A, \Delta_B$  are the IT-MAC keys of  $P_A, P_B$  respectively.*

We remark that for any  $x \in \mathbb{F}_{2^\kappa}$  the IT-MAC authentication  $[x\Delta_A]_{\Delta_B}$  can be locally transformed to  $\langle x \rangle$ , which we summarize in the following macro (the case for  $[\Delta_B]_{\Delta_A}$  can be defined analogously). In particular, by computing  $[\Delta_B]_{\Delta_A}$  we implicitly have  $\langle 1 \rangle$ , i.e., authentication of the constant  $1 \in \mathbb{F}_{2^\kappa}$ .

–  $\langle x \rangle \leftarrow \text{Convert1}_{[\cdot] \rightarrow \langle \cdot \rangle}([x\Delta_B]_{\Delta_A})$ : Set  $D_A[x] := M_A[x\Delta_B]$  and  $D_B[x] := K_B[x\Delta_B]$ .

For the ease of presentation, we also define the following macro that generates dual key authentication of cross terms  $\langle xy \rangle$  assuming the existence of  $\langle y \rangle := (\alpha, \beta)$  and  $[x]_{A,\beta} = (K_B[x]_\beta, M_A[x]_\beta, x)$ . The correctness can be verified straightforwardly.

–  $\langle xy \rangle \leftarrow \text{Convert2}_{[\cdot] \rightarrow \langle \cdot \rangle}([x]_{A,\beta}, \langle y \rangle)$ : Given IT-MAC  $[x]_{A,\beta}$  and dual-key authentication  $\langle y \rangle$ ,  $P_A$  and  $P_B$  locally compute the following steps:

**Functionality  $\mathcal{F}_{\text{cpre}}$** 

This functionality is parameterized by a Boolean circuit  $\mathcal{C}$  consisting of a list of gates in the form of  $(i, j, k, T)$ . Let  $n := |\mathcal{W}| + |\mathcal{I}_B|$  (resp.,  $m := |\mathcal{W}| + |\mathcal{I}_A|$ ) be the number of all AND gates as well as circuit-input gates corresponding to the input of  $\mathsf{P}_B$  (resp.,  $\mathsf{P}_A$ ), and  $L = \lceil \rho \log \frac{2en}{\rho} + \frac{\log \rho}{2} \rceil$  be a compression parameter. It runs with parties  $\mathsf{P}_A, \mathsf{P}_B$  and the ideal-world adversary  $\mathcal{S}$ , and operates as follows:

**Initialize.** Sample two global keys  $\Delta_A, \Delta_B \in \mathbb{F}_{2^\kappa}$  as follows:

- If  $\mathsf{P}_A$  is honest, sample  $\Delta_A \leftarrow \mathbb{F}_{2^\kappa}$  such that  $\text{lsb}(\Delta_A) = 1$ . Otherwise, receive  $\Delta_A \in \mathbb{F}_{2^\kappa}$  with  $\text{lsb}(\Delta_A) = 1$  from  $\mathcal{S}$ .
- If  $\mathsf{P}_B$  is honest, sample  $\Delta_B \leftarrow \mathbb{F}_{2^\kappa}$  such that  $\text{lsb}(\Delta_A \Delta_B) = 1$  and  $\text{msb}(\Delta_B) = 1$ . Otherwise, receive  $\Delta_B \in \mathbb{F}_{2^\kappa}$  with  $\text{msb}(\Delta_B) = 1$  from  $\mathcal{S}$ , and then re-sample  $\Delta_A \leftarrow \mathbb{F}_{2^\kappa}$  such that  $\text{lsb}(\Delta_A \Delta_B) = 1$  and  $\text{lsb}(\Delta_A) = 1$ .
- Store  $(\Delta_A, \Delta_B)$ , and output  $\Delta_A$  and  $\Delta_B$  to  $\mathsf{P}_A$  and  $\mathsf{P}_B$ , respectively.

**Macro.  $\text{Auth}_A(\mathbf{x}, \ell)$**  (this is an internal subroutine only)

- If  $\mathsf{P}_B$  is honest, sample  $\mathsf{K}_B[\mathbf{x}] \leftarrow \mathbb{F}_{2^\kappa}^\ell$ ; otherwise, receive  $\mathsf{K}_B[\mathbf{x}] \in \mathbb{F}_{2^\kappa}^\ell$  from  $\mathcal{S}$ .
- If  $\mathsf{P}_A$  is honest, compute  $\mathsf{M}_A[\mathbf{x}] := \mathsf{K}_B[\mathbf{x}] + \mathbf{x} \cdot \Delta_B \in \mathbb{F}_{2^\kappa}^\ell$ . Otherwise, receive  $\mathsf{M}_A[\mathbf{x}] \in \mathbb{F}_{2^\kappa}^\ell$  from  $\mathcal{S}$ , and recompute  $\mathsf{K}_B[\mathbf{x}] := \mathsf{M}_A[\mathbf{x}] + \mathbf{x} \cdot \Delta_B \in \mathbb{F}_{2^\kappa}^\ell$ .
- Send  $(\mathbf{x}, \mathsf{M}_A[\mathbf{x}])$  to  $\mathsf{P}_A$  and  $\mathsf{K}_B[\mathbf{x}]$  to  $\mathsf{P}_B$ .

$\text{Auth}_B(\mathbf{x}, \ell)$  can be defined similarly by swapping the roles of  $\mathsf{P}_A$  and  $\mathsf{P}_B$ .

**Preprocess the circuit with compressed wire masks.** Sample  $\mathsf{M} \leftarrow \mathbb{F}_2^{n \times L}$ , and then execute as follows:

- For  $w \in \mathcal{I}_A$ , set  $b_w = 0$  and define  $[b_w]$ ; for  $w \in \mathcal{I}_B$ , set  $a_w = 0$  and define  $[a_w]$ .
- If  $\mathsf{P}_A$  is honest, sample  $\mathbf{a} \leftarrow \mathbb{F}_2^m$ ; otherwise, receive  $\mathbf{a} \in \mathbb{F}_2^m$  from  $\mathcal{S}$ . Then, execute  $\text{Auth}_A(\mathbf{a}, m)$  to generate  $[\mathbf{a}]$ . For each wire  $w \in \mathcal{I}_A \cup \mathcal{W}$ , define  $a_w$  as the wire mask held by  $\mathsf{P}_A$ .
- If  $\mathsf{P}_B$  is honest, sample  $\mathbf{b}^* \leftarrow \mathbb{F}_2^L$ ; otherwise, receive  $\mathbf{b}^* \in \mathbb{F}_2^L$  from  $\mathcal{S}$ . Run  $\text{Auth}_B(\mathbf{b}^*, L)$  to generate  $[\mathbf{b}^*]$ , and then compute  $[\mathbf{b}] := \mathsf{M} \cdot [\mathbf{b}^*]$  with  $\mathbf{b} \in \mathbb{F}_2^n$ . For each wire  $w \in \mathcal{I}_B \cup \mathcal{W}$ , define  $b_w$  as the wire mask held by  $\mathsf{P}_B$ .
- In a topological order, for each gate  $(i, j, k, T)$ , do the following:
  - If  $T = \oplus$ , compute  $[a_k] := [a_i] \oplus [a_j]$  and  $[b_k] := [b_i] \oplus [b_j]$ .
  - If  $T = \wedge$ , execute as follows:
    1. If  $\mathsf{P}_A$  is honest, then sample  $\hat{a}_k \leftarrow \{0, 1\}$ , else receive  $\hat{a}_k \in \{0, 1\}$  from  $\mathcal{S}$ .
    2. If  $\mathsf{P}_B$  is honest, then compute  $\hat{b}_k := (a_i \oplus b_i) \wedge (a_j \oplus b_j) \oplus \hat{a}_k$ . Otherwise, receive  $\hat{b}_k \in \{0, 1\}$  from  $\mathcal{S}$ , and re-compute  $\hat{a}_k := (a_i \oplus b_i) \wedge (a_j \oplus b_j) \oplus \hat{b}_k$ .

Let  $\hat{\mathbf{a}}$  and  $\hat{\mathbf{b}}$  be the vectors consisting of bits  $\hat{a}_k$  and  $\hat{b}_k$  for  $k \in \mathcal{W}$ . Run  $\text{Auth}_A(\hat{\mathbf{a}})$  and  $\text{Auth}_B(\hat{\mathbf{b}})$  to generate  $[\hat{\mathbf{a}}]$  and  $[\hat{\mathbf{b}}]$ , respectively.

- Output  $\mathsf{M}$  and  $([\mathbf{a}], [\hat{\mathbf{a}}], [\mathbf{b}^*], [\hat{\mathbf{b}}])$  to  $\mathsf{P}_A$  and  $\mathsf{P}_B$ .

**Fig. 3.** Compressed preprocessing functionality for authenticated triples.

- $\mathsf{P}_A$  outputs  $\mathsf{D}_A[xy] := \alpha \cdot x + \mathsf{M}_A[x]_\beta \in \mathbb{F}_{2^\kappa}$ .
- $\mathsf{P}_B$  outputs  $\mathsf{D}_B[xy] := \mathsf{K}_B[x]_\beta$ .

In our protocol we utilize the following properties of dual key authentication. Since they are straightforward we only provide brief explanation and refrain from providing detailed description.

*Claim.* The dual-key authentication is additively homomorphic. In particular, given  $\langle x_1 \rangle := (\mathsf{D}_A[x_1], \mathsf{D}_B[x_1])$  and  $\langle x_2 \rangle := (\mathsf{D}_A[x_2], \mathsf{D}_B[x_2])$ ,  $\mathsf{P}_A, \mathsf{P}_B$  can locally compute  $\langle x_1 + x_2 \rangle := (\mathsf{D}_A[x_1] + \mathsf{D}_A[x_2], \mathsf{D}_B[x_1] + \mathsf{D}_B[x_2])$ .

The additive homomorphism of dual-key authentication implies that given public coefficients  $c_0, c_1, \dots, c_\ell \in \mathbb{F}_{2^\kappa}$ , two parties can *locally* compute  $\langle y \rangle := c_0 + \sum_{i=1}^{\ell} c_i \cdot \langle x_i \rangle$ .

We define the zero-checking macro `CheckZero2` which ensures soundness for both parties. We note that this is simply the equality checking operations.

- `CheckZero2`( $\langle x_1 \rangle, \dots, \langle x_\ell \rangle$ ): On input dual-key authenticated values  $\langle x_1 \rangle, \dots, \langle x_\ell \rangle$  both parties check  $x_i = 0$  for  $i \in [1, \ell]$  as follows:
  1.  $\mathsf{P}_A$  computes  $h_A := \mathsf{H}(\mathsf{D}_A[x_1], \dots, \mathsf{D}_A[x_\ell])$ , and  $\mathsf{P}_B$  sets  $h_B := \mathsf{H}(\mathsf{D}_B[x_1], \dots, \mathsf{D}_B[x_\ell])$ , where  $\mathsf{H} : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa$  is a random oracle.
  2. Both parties call functionality  $\mathcal{F}_{\text{EQ}}$  to check  $h_A = h_B$ . If  $\mathcal{F}_{\text{EQ}}$  outputs `false`, the parties abort.

Notice that the additive homomorphic and zero-checking properties allow us to check that a dual-key authenticated value  $\langle x \rangle$  matches a public value  $x'$  assuming the existence of  $\langle 1 \rangle = (\mathsf{D}_A[1], \mathsf{D}_B[1])$  by calling `CheckZero2`( $\langle x \rangle - x' \langle 1 \rangle$ ). Similar to `CheckZero` we have the following soundness lemma of `CheckZero2`.

**Lemma 2.** *If  $\Delta_A, \Delta_B \in \mathbb{F}_{2^\kappa}$  is sampled uniformly at random and are non-zero, then the probability that there exists some  $i \in [1, \ell]$  such that  $x_i \neq 0$  and  $\mathsf{P}_A$  or  $\mathsf{P}_B$  accepts in the `CheckZero2` procedure is bounded by  $\frac{2}{2^\kappa}$ .*

## 4.2 Global-Key Sampling

We require  $\Delta_A \neq 0$ ,  $\Delta_B \neq 0$ , and  $\text{lsb}(\Delta_A \Delta_B) = 1$  in the preprocessing phase to facilitate dual-key authentication. Considering the requirement of half-gates garbling, we have the constraints  $\text{lsb}(\Delta_A) = 1$ ,  $\text{msb}(\Delta_B) = 1$ , and  $\text{lsb}(\Delta_A \Delta_B) = 1$  in  $\mathcal{F}_{\text{cpre}}$ . We design the protocol  $\Pi_{\text{samp}}$  in Fig. 4 and argue in Lemma 3 that the key constraints are satisfied.

**Lemma 3.** *The protocol  $\Pi_{\text{samp}}$  satisfies the following properties:*

- *The outputs satisfy that  $\text{lsb}(\Delta_A) = 1$ ,  $\text{msb}(\Delta_B) = 1$ , and  $\text{lsb}(\Delta_A \Delta_B) = 1$  in the honest case.*
- *If  $\text{lsb}(\Delta_A) \neq 1$  then  $\mathsf{P}_B$  aborts except with probability  $2^{-\rho}$ . Conditioned on  $\Delta_A \neq 0$ , if  $\text{lsb}(\Delta_A \Delta_B) \neq 1$  then  $\mathsf{P}_B$  aborts except with probability  $2^{-\rho}$ .*
- *If  $\text{msb}(\Delta_B) \neq 1$  then  $\mathsf{P}_A$  aborts except with probability  $2^{-\rho}$ . Conditioned on  $\Delta_B \neq 0$ , if  $\text{lsb}(\Delta_A \Delta_B) \neq 1$  then  $\mathsf{P}_B$  aborts except with probability  $2 \cdot 2^{-\kappa} + 2^{-\rho}$ .*

Protocol  $\Pi_{\text{samp}}$ 

$\mathsf{P}_A$  samples  $\Delta_A \leftarrow \mathbb{F}_{2^\kappa}$  such that  $\text{lsb}(\Delta_A) = 1$ .  $\mathsf{P}_B$  samples  $\tilde{\Delta}_B \leftarrow \mathbb{F}_{2^\kappa}$  such that  $\text{msb}(\tilde{\Delta}_B) = 1$ . Then,  $\mathsf{P}_A$  and  $\mathsf{P}_B$  execute the following steps.

1.  $\mathsf{P}_A$  and  $\mathsf{P}_B$  call functionality  $\mathcal{F}_{\text{COT}}$  on respective input  $(\text{init}, \text{sid}_0, \Delta_A)$  and  $(\text{init}, \text{sid}_0)$ , and then call  $\mathcal{F}_{\text{COT}}$  on the same input  $(\text{extend}, \text{sid}_0, \rho)$  to generate random authenticated bits  $[u]_B$ .
2. Then  $\mathsf{P}_A$  convinces  $\mathsf{P}_B$  that  $\text{lsb}(\Delta_A) = 1$  by sending a  $\rho$ -bit vector  $m_A^0 := (\text{lsb}(\mathsf{K}_A[u_1]), \dots, \text{lsb}(\mathsf{K}_A[u_\rho]))$  to  $\mathsf{P}_B$ , who checks that  $m_A^0 = (\text{lsb}(\mathsf{M}_B[u_1]) \oplus u_1, \dots, \text{lsb}(\mathsf{M}_B[u_\rho]) \oplus u_\rho)$  holds.
3.  $\mathsf{P}_B$  runs  $\text{Fix}(\text{sid}_0, \tilde{\Delta}_B)$  to generate  $[\tilde{\Delta}_B]_{\Delta_A}$ . Then,  $\mathsf{P}_A$  sends  $m_A^1 = \text{lsb}(\mathsf{K}_A[\tilde{\Delta}_B])$  to  $\mathsf{P}_B$ , and  $\mathsf{P}_B$  sends  $m_B^1 = \text{lsb}(\mathsf{M}_B[\tilde{\Delta}_B])$  to  $\mathsf{P}_A$  in parallel. If  $m_A^1 \oplus m_B^1 = 0$ , both parties compute  $[\Delta_B]_{\Delta_A} := [\tilde{\Delta}_B]_{\Delta_A} \oplus 1$  where  $\Delta_B = \tilde{\Delta}_B \oplus 1$ ; otherwise, the parties set  $[\Delta_B]_{\Delta_A} := [\tilde{\Delta}_B]_{\Delta_A}$ .
4.  $\mathsf{P}_A$  and  $\mathsf{P}_B$  call  $\mathcal{F}_{\text{COT}}$  on respective input  $(\text{init}, \text{sid}'_0)$  and  $(\text{init}, \text{sid}'_0, \Delta_B)$ , and then call  $\mathcal{F}_{\text{COT}}$  on the same input  $(\text{extend}, \text{sid}'_0, \rho)$  to generate random authenticated bits  $[v]_A$ .
5. Then  $\mathsf{P}_B$  convinces  $\mathsf{P}_A$  that  $\text{msb}(\Delta_B) = 1$  by sending a  $\rho$ -bit vector  $m_B^0 := (\text{msb}(\mathsf{K}_B[v_1]), \dots, \text{msb}(\mathsf{K}_B[v_\rho]))$  to  $\mathsf{P}_A$ , who checks that  $m_B^0 = (\text{msb}(\mathsf{M}_A[v_1]) \oplus v_1, \dots, \text{msb}(\mathsf{M}_A[v_\rho]) \oplus v_\rho)$  holds.
6.  $\mathsf{P}_A$  and  $\mathsf{P}_B$  execute the following steps to mutually check that  $\text{lsb}(\Delta_A \cdot \Delta_B) = 1$ .
  - (a) Both parties call  $\mathcal{F}_{\text{COT}}$  on the same input  $(\text{extend}, \text{sid}_0, \rho)$  to generate random authenticated bits  $[x]_B$ , as well as run  $\text{Fix}(\text{sid}_0, \Delta_B \cdot x)$  to generate  $[\Delta_B \cdot x]_B$ .  $\mathsf{P}_B$  proves to  $\mathsf{P}_A$  that a set of authenticated triples  $\{([x_i]_B, [\Delta_B]_B, [x_i \Delta_B]_B)\}_{i \in [1, \rho]}$  is valid by calling  $\mathcal{F}_{\text{DVZK}}$ , and  $\mathsf{P}_A$  aborts if it receives **false** from  $\mathcal{F}_{\text{DVZK}}$ .
  - (b) Both parties set  $\langle x \rangle := \text{Convert}_{[1] \rightarrow \langle \cdot \rangle}([\Delta_B \cdot x]_B)$ . Then,  $\mathsf{P}_A$  sends  $m_A^2 := (\text{lsb}(\mathsf{D}_A[x_1]), \dots, \text{lsb}(\mathsf{D}_A[x_\rho]))$  to  $\mathsf{P}_B$ , who checks that  $m_A^2 = (\text{lsb}(\mathsf{D}_B[x_1]) \oplus x_1, \dots, \text{lsb}(\mathsf{D}_B[x_\rho]) \oplus x_\rho)$ .
  - (c) The parties run  $\text{Fix}(\text{sid}'_0, \Delta_A)$  to generate  $[\Delta_A]_A$ .
  - (d) Both parties call  $\mathcal{F}_{\text{COT}}$  on the same input  $(\text{extend}, \text{sid}'_0, \rho)$  to generate random authenticated bits  $[y]_A$ , as well as run  $\text{Fix}(\text{sid}'_0, \Delta_A \cdot y)$  to generate  $[\Delta_A \cdot y]_A$ .  $\mathsf{P}_B$  proves to  $\mathsf{P}_A$  that a set of authenticated triples  $\{([y_i]_A, [\Delta_A]_A, [y_i \Delta_A]_A)\}_{i \in [1, \rho]}$  is valid by calling  $\mathcal{F}_{\text{DVZK}}$ , and  $\mathsf{P}_B$  aborts if it receives **false** from  $\mathcal{F}_{\text{DVZK}}$ .
  - (e) Both parties set  $\langle y \rangle := \text{Convert}_{[1] \rightarrow \langle \cdot \rangle}([\Delta_A \cdot y]_A)$ . Then,  $\mathsf{P}_B$  sends  $m_B^2 := (\text{lsb}(\mathsf{D}_B[y_1]), \dots, \text{lsb}(\mathsf{D}_B[y_\rho]))$  to  $\mathsf{P}_A$ , who checks that  $m_B^2 = (\text{lsb}(\mathsf{D}_A[y_1]) \oplus y_1, \dots, \text{lsb}(\mathsf{D}_A[y_\rho]) \oplus y_\rho)$ .
  - (f) Both parties locally compute two dual-key authenticated bits  $\langle 1_B \rangle := \text{Convert}_{[1] \rightarrow \langle \cdot \rangle}([\Delta_B]_B)$  and  $\langle 1_A \rangle := \text{Convert}_{[1] \rightarrow \langle \cdot \rangle}([\Delta_A]_A)$ .
  - (g) The parties run  $\text{CheckZero2}(\langle 1_B \rangle - \langle 1_A \rangle)$ , and abort if the check fails.
7.  $\mathsf{P}_A$  outputs  $(\Delta_A, \alpha_0)$  and  $\mathsf{P}_B$  outputs  $(\Delta_B, \beta_0)$ , such that  $\text{lsb}(\Delta_A) = 1$ ,  $\text{msb}(\Delta_B) = 1$ ,  $\text{lsb}(\Delta_A \cdot \Delta_B) = 1$  and  $\alpha_0 + \beta_0 = \Delta_A \cdot \Delta_B \in \mathbb{F}_{2^\kappa}$ .

**Fig. 4.** Sub-protocol for sampling global keys.

*Proof.* For the honest case since  $P_A$  and  $P_B$  follow the protocol instruction when sampling keys, the constraints on  $\Delta_A$  and  $\Delta_B$  are satisfied automatically. Moreover, notice that  $\text{lsb}(\Delta_A \tilde{\Delta}_B) = \text{lsb}(K_A[\tilde{\Delta}_B]) \oplus \text{lsb}(M_B[\tilde{\Delta}_B])$  and  $\text{lsb}(\Delta_A) = 1$ . If the parties discover in step 6b that  $\text{lsb}(\Delta_A \tilde{\Delta}_B) = 0$ ,  $P_B$  sets  $\Delta_B := \tilde{\Delta}'_B \oplus 1$  and  $\text{lsb}(\Delta_A \Delta_B) = \text{lsb}(\Delta_A \tilde{\Delta}_B + \Delta_A) = 1$ .

For the case of a corrupted  $P_A$ , notice that  $\text{lsb}(K_A[r]) \oplus \text{lsb}(M_B[r]) = r \cdot \text{lsb}(\Delta_A)$  and  $\text{lsb}(D_A[r]) \oplus \text{lsb}(D_B[r]) = r \cdot \text{lsb}(\Delta_A \Delta_B)$  for  $r \in \mathbb{F}_2$ . If  $\text{lsb}(\Delta_A) = 0$  then  $P_A$  passing the test is equivalent to  $m_A^0 \oplus (\text{lsb}(K_A[u_1]), \dots, \text{lsb}(K_A[u_\rho])) = \mathbf{u}$  which happens with  $2^{-\rho}$  probability since  $\mathbf{u}$  is sampled independently from the left-hand side of the equation. Conditioned on  $\Delta_A \neq 0$ , the second test passes when  $\text{lsb}(\Delta_A \Delta_B) = 0$  except with  $2^{-\rho}$  probability from similar argument.

For the case of a corrupted  $P_B$ , the checks in step 5 and step 6e are equivalent to the corrupted  $P_A$  case. Thus the soundness of the first check is  $2^{-\rho}$ . Also Lemma 2 guarantees that inconsistent  $\Delta_B$  will be detected except with  $2 \cdot 2^{-\kappa}$  probability. By union bound the soundness of the second check is  $2 \cdot 2^{-\kappa} + 2^{-\rho}$ .

### 4.3 Consistency Check Between Values and MAC Tags

In our protocol to generate dual-key authentication, we need a party (e.g.,  $P_B$ ) to use the MAC tags (denoted as  $\{\beta_i\}$ ) of some existing IT-MAC authenticated values as the global keys of another  $\mathcal{F}_{\text{bCOT}}$  instance (denoted as  $\{\beta'_i\}$ ). We enforce this constraint by checking equality between values authenticated by different keys. Our first observation is that the MAC tags are already implicitly authenticated by  $\Delta_A^{-1}$ .

*Authentication Under Inverse Key.* We define the `Invert` macro to *locally* convert  $[x]_B = (K_A[x], M_B[x], x)$  to  $[y]_{B, \Delta_A^{-1}} := (K_A[y]_{\Delta_A^{-1}}, M_B[y]_{\Delta_A^{-1}}, y)$ . We note that this technique appeared previously in the certified VOLE protocols [18].

- $[y]_{B, \Delta_A^{-1}} \leftarrow \text{Invert}([x]_B)$ : On input  $[x]_B$  for  $x \in \mathbb{F}_{2^\kappa}$ ,  $P_A$  and  $P_B$  execute the following:
  - $P_B$  outputs  $y := M_B[x]$  and  $M_B[y]_{\Delta_A^{-1}} := x$ .
  - $P_A$  outputs  $K_A[y]_{\Delta_A^{-1}} := K_A[x] \cdot \Delta_A^{-1} \in \mathbb{F}_{2^\kappa}$ .

We demonstrate the correctness of the `Invert` macro as follows.

**Lemma 4.** *Let  $[x]_B = (\alpha, \beta, x)$  where  $x \in \mathbb{F}_{2^\kappa}$  then the MAC tag of  $P_B$ ,  $\beta$ , is implicitly authenticated by  $\Delta_A^{-1}$ , i.e., the inverse of  $P_A$ 's global key over  $\mathbb{F}_{2^\kappa}$ .*

This claim can be verified by multiplying both side of the equation by  $\Delta_A^{-1}$ .

$$\underbrace{\beta}_{M_B[x]} = \underbrace{\alpha}_{K_A[x]} + x \cdot \Delta_A \implies \underbrace{x}_{M_B[\beta]_{\Delta_A^{-1}}} = \underbrace{\alpha \cdot \Delta_A^{-1}}_{K_A[\beta]_{\Delta_A^{-1}}} + \beta \cdot \Delta_A^{-1} .$$

*Random Inverse Key Authentication.* Notice that in the `Invert` macro, if we require the input  $[x]$  to be uniformly random, i.e.,  $x \leftarrow \mathbb{F}_{2^\kappa}$ , then the output value  $y := \mathsf{M}_A[x] = x\Delta_A - \mathsf{K}_B[x]$  is also uniformly random in the view of  $\mathsf{P}_A$ . Using this method we can generate random  $\mathbb{F}_{2^\kappa}$  elements authenticated by  $\Delta_A^{-1}$ .

*Equality Check Across Different Keys.* We recall a known technique to verify equality between two values authenticated by respective independent keys [16], which we summarize in the `EQCheck` macro. We recall its soundness in Lemma 5 and prove it in the full version [14]. In the following, we assume that  $\mathcal{F}_{\text{COT}}$  has been initialized with  $(\textit{sid}, \Delta_A)$  and  $(\textit{sid}', \Delta'_A)$ .

- `EQCheck`( $\{[y_i]_{\Delta_A}\}_{i \in [1, \ell]}, \{[y'_i]_{\Delta'_A}\}_{i \in [1, \ell]}$ ): On input two sets of authenticated values under different keys  $\Delta_A, \Delta'_A$ ,  $\mathsf{P}_A$  and  $\mathsf{P}_B$  check that  $y_i = y'_i$  for all  $i \in [1, \ell]$  as follows:
  1. Let  $[y_i]_{\Delta_A} = (k_i, m_i, y_i)$  and  $[y'_i]_{\Delta'_A} = (k'_i, m'_i, y'_i)$ . Two parties  $\mathsf{P}_A$  and  $\mathsf{P}_B$  run `Fix`( $\textit{sid}, \{m'_i\}_{i \in [1, \ell]}$ ) to obtain a set of authenticated values  $\{[m'_i]_{\Delta_A}\}_{i \in [1, \ell]}$ , and also run `Fix`( $\textit{sid}', \{m_i\}_{i \in [1, \ell]}$ ) to get another set of authenticated values  $\{[m_i]_{\Delta'_A}\}_{i \in [1, \ell]}$ .
  2. For each  $i \in [1, \ell]$ ,  $\mathsf{P}_A$  computes  $V_i := k_i \cdot \Delta'_A + k'_i \cdot \Delta_A + \mathsf{K}_A[m_i]_{\Delta'_A} + \mathsf{K}_A[m'_i]_{\Delta_A} \in \mathbb{F}_{2^\kappa}$ , and  $\mathsf{P}_B$  computes  $W_i := \mathsf{M}_B[m_i]_{\Delta'_A} + \mathsf{M}_B[m'_i]_{\Delta_A} \in \mathbb{F}_{2^\kappa}$ .
  3.  $\mathsf{P}_B$  sends  $h := \mathsf{H}(W_1, \dots, W_\ell)$  to  $\mathsf{P}_A$ , who verifies that  $h = \mathsf{H}(V_1, \dots, V_\ell)$ . If the check fails,  $\mathsf{P}_A$  aborts.

**Lemma 5.** *If  $\Delta_A$  and  $\Delta'_A$  are independently sampled from  $\mathbb{F}_{2^\kappa}$ , then the probability that there exists some  $i \in [1, \ell]$  such that  $y_i \neq y'_i$  and  $\mathsf{P}_A$  accepts in the `EQCheck` procedure is bounded by  $\frac{3}{2^\kappa}$ .*

*The Consistency Check.* The observation in Lemma 4 suggests that the MAC tags  $\{\beta_i\}$  are already implicitly authenticated by  $\Delta_A^{-1}$ . Moreover, by calling `Fix`( $\Delta'_A$ ),  $\mathsf{P}_A$  and  $\mathsf{P}_B$  can acquire  $\{[\Delta'_A]_{\beta'_i}\}$  and locally convert them to  $\{[\beta'_i]_{\Delta'_A}\}$ . Since  $\Delta_A$  and  $\Delta'_A$  are independent, we can apply `EQCheck` to complete our goal.

We list the differences that inverse key authentication induces to `EQCheck`. Recall that  $\mathcal{F}_{\text{COT}}$  has been initialized with  $(\textit{sid}, \Delta_A)$  and  $(\textit{sid}', \Delta'_A)$ .

- `EQCheck`( $\{[\beta_i]_{\Delta_A^{-1}}\}_{i \in [1, \ell]}, \{[\beta'_i]_{\Delta'_A}\}_{i \in [1, \ell]}$ ): On input two sets of authenticated values under different keys  $\Delta_A^{-1}, \Delta'_A$ ,  $\mathsf{P}_A$  and  $\mathsf{P}_B$  check that  $\beta_i = \beta'_i$  for all  $i \in [1, \ell]$  as follows:
  1.  $\mathsf{P}_A$  and  $\mathsf{P}_B$  call  $\mathcal{F}_{\text{COT}}$  on the same input (`extend`,  $\textit{sid}$ ,  $\ell\kappa$ ) to get authenticated bits  $[r_1]_{\Delta_A}, \dots, [r_\ell]_{\Delta_A}$  with  $r_i \in \mathbb{F}_2^\kappa$ . Then, for  $i \in [1, \ell]$ , both parties define  $[r_i]_{\Delta_A} := \mathsf{B2F}([r_i]_{\Delta_A})$  with  $r_i \in \mathbb{F}_{2^\kappa}$ , and set  $[s_i]_{\Delta_A^{-1}} := \mathsf{Invert}([r_i]_{\Delta_A})$ .
  2.  $\mathsf{P}_A$  and  $\mathsf{P}_B$  run `EQCheck`( $\{[\beta_i]_{\Delta_A^{-1}}\}_{i \in [1, \ell]}, \{[\beta'_i]_{\Delta'_A}\}_{i \in [1, \ell]}$ ) as described above, except that they use random authenticated values  $[s_i]_{\Delta_A^{-1}}$  for  $i \in [1, \ell]$  to generate chosen authenticated values under  $\Delta_A^{-1}$  in the `Fix` procedure.



It is straightforward to verify the soundness is not affected by changing to the inverse key. Thus we omit the proof of the following lemma.

**Lemma 6.** *If  $\Delta_A$  and  $\Delta'_A$  are independently sampled from  $\mathbb{F}_{2^\kappa}$ , then the probability that there exists some  $i \in [1, \ell]$  such that  $\beta_i \neq \beta'_i$  and  $P_A$  accepts in the EQCheck procedure is bounded by  $\frac{3}{2^\kappa}$ .*

#### 4.4 Circuit Dependent Compressed Preprocessing

We now describe the protocol to realize the functionality  $\mathcal{F}_{\text{cpre}}$ . Following the conventions of previous works, we defer all consistency checks to the end of the protocol. Notice that step 1 to step 5 corresponds to the circuit-independent phase (where we only require the scale rather than the topology information of the circuit) while the rest is the circuit-dependent phase (where the entire circuit is known). The protocol is shown in Fig. 5 and Fig. 6. We then analyze its security in Theorem 1. The proof is presented in the full version [14].

**Theorem 1.** *Protocol  $\Pi_{\text{cpre}}$  shown in Figs. 5 and 6 securely realizes functionality  $\mathcal{F}_{\text{cpre}}$  (Fig. 3) against malicious adversaries in the  $(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\text{bcOT}}, \mathcal{F}_{\text{DVZK}}, \mathcal{F}_{\text{EQ}}, \mathcal{F}_{\text{Rand}})$ -hybrid model.*

*Consistency Checks.* We explain the rationale of the consistency checks in  $\Pi_{\text{cpre}}$ .

- The  $\mathcal{F}_{\text{DVZK}}$  in step 11 checks that the Fix inputs of  $P_A$  in step 6 and those of  $P_B$  in step 6 and step 3 are well-formed.
- The CheckZero2 and EQCheck in step 12 ensure to  $P_A$  that the multiple instances of  $\Delta_B$  in  $\Pi_{\text{samp}}$  (Fig. 4) and  $\Pi_{\text{cpre}}$  (step 4 and step 5 in Fig. 5) are identical. Also,  $P_B$  can make sure that  $\Delta'_A$  in step 4 and step 5 of  $\Pi_{\text{cpre}}$  (Fig. 5) are identical.
- $P_B$  checks that the message in step 9 of  $\Pi_{\text{cpre}}$  from  $P_A$  are correct. To do this,  $P_B$  checks its locally computed value against the dual-key authenticated value, which is unalterable. Moreover, we reduce the communication using random linear combination. This is done in step 14 and step 15 of  $\Pi_{\text{cpre}}$  (Fig. 6).
- $P_A$  checks that the Fix inputs of  $P_B$  in step 10 of  $\Pi_{\text{cpre}}$  (Fig. 6) are correct. This is done by checking the IT-MAC authenticated values against the dual-key authenticated ones in step 16 of  $\Pi_{\text{cpre}}$  (Fig. 6).

*Optimization Based on Fiat-Shamir.* In the protocol  $\Pi_{\text{cpre}}$ , both parties choose random public challenges by calling functionality  $\mathcal{F}_{\text{Rand}}$ . Based on the Fiat-Shamir heuristic [19], both parties can generate the challenges by hashing the protocol transcript up until this point, which is secure in the random oracle model. This optimization can save one communication round, and has also been used in previous work such as [10, 43].

**Protocol  $\Pi_{\text{cpre}}$**

**Inputs:** A Boolean circuit  $\mathcal{C}$  that consists of a list of gates of the form  $(i, j, k, T)$ . Let  $n = |\mathcal{W}| + |\mathcal{I}_B|$ ,  $m = |\mathcal{W}| + |\mathcal{I}_A|$ ,  $L = \lceil \rho \log \frac{2en}{\rho} + \frac{\log \rho}{2} \rceil$  and  $t = |\mathcal{W}|$ .

**Initialize:**  $P_A$  and  $P_B$  execute sub-protocol  $\Pi_{\text{samp}}$  (Figure 4) to obtain  $(\Delta_A, \alpha_0)$  and  $(\Delta_B, \beta_0)$  respectively, such that  $\text{lsb}(\Delta_A) = 1$ ,  $\text{msb}(\Delta_B) = 1$ ,  $\text{lsb}(\Delta_A \cdot \Delta_B) = 1$  and  $\alpha_0 + \beta_0 = \Delta_A \cdot \Delta_B \in \mathbb{F}_{2^\kappa}$ . Thus, both parties hold  $\langle 1 \rangle$  (i.e.,  $[\Delta_B]_{\Delta_A}$ ). After the sub-protocol execution,  $\mathcal{F}_{\text{COT}}$  was initialized by session identifier  $\text{sid}_0$  and  $\Delta_A$ .

**Generate authenticated AND triples:**  $P_A$  and  $P_B$  execute as follows:

1.  $P_B$  samples a matrix  $\mathbf{M} \leftarrow \mathbb{F}_2^{n \times L}$  and sends it to  $P_A$ .
2. Both parties call  $\mathcal{F}_{\text{COT}}$  on input  $(\text{extend}, \text{sid}_0, L)$  to generate random authenticated bits  $[b^*]$  where  $b^* \in \mathbb{F}_2^L$  and compute  $[b] := \mathbf{M} \cdot [b^*]$  with  $b \in \mathbb{F}_2^n$ .
3. Both parties run  $\text{Fix}(\text{sid}_0, \{b_i^* \Delta_B\}_{i \in [1, L]})$  to generate authenticated values  $[b_i^* \Delta_B]_B$ . The parties locally run  $\langle b_i^* \rangle \leftarrow \text{Convert1}_{[1] \rightarrow \langle \cdot \rangle}([b_i^* \Delta_B]_{\Delta_A})$ . Let  $\alpha_i, \beta_i \in \mathbb{F}_{2^\kappa}$  such that  $\alpha_i + \beta_i = b_i^* \cdot \Delta_A \cdot \Delta_B$  for each  $i \in [1, L]$ .
4.  $P_B$  and  $P_A$  call  $\mathcal{F}_{\text{COT}}^{L+1}$  on respective inputs  $(\text{init}, \text{sid}_1, \beta_1, \dots, \beta_L, \Delta_B)$  and  $(\text{init}, \text{sid}_1)$ . Then, both parties send  $(\text{extend}, \text{sid}_1, m)$  to  $\mathcal{F}_{\text{COT}}^{L+1}$ , which returns  $([a]_{\beta_1}, \dots, [a]_{\beta_L}, [a]_{\Delta_A})$  where  $a \in \mathbb{F}_2^m$ . Then,  $P_A$  samples  $\Delta'_A \leftarrow \mathbb{F}_{2^\kappa}$ , and then two parties run  $\text{Fix}(\text{sid}_1, \Delta'_A)$  to obtain  $([\Delta'_A]_{\beta_1}, \dots, [\Delta'_A]_{\beta_L}, [\Delta'_A]_{\Delta_B})$ .  $P_A$  and  $P_B$  set  $\langle 1_B^{(1)} \rangle := \text{Convert1}_{[1] \rightarrow \langle \cdot \rangle}([\Delta_B]_{\Delta'_A})$  where  $[\Delta_B]_{\Delta'_A}$  is equivalent to  $[\Delta'_A]_{\Delta_B}$ , and define  $[\beta_i]_{\Delta'_A} = [\Delta'_A]_{\beta_i}$  for  $i \in [1, L]$ .
5.  $P_B$  and  $P_A$  call  $\mathcal{F}_{\text{COT}}^2$  on respective input  $(\text{init}, \text{sid}_2, \beta_0, \Delta_B)$  and  $(\text{init}, \text{sid}_2)$ . Then, both parties send  $(\text{extend}, \text{sid}_2, t)$  to  $\mathcal{F}_{\text{COT}}^2$ , which returns  $([\hat{a}]_{\beta_0}, [\hat{a}]_{\Delta_B})$  to the parties.  $P_A$  and  $P_B$  run  $\text{Fix}(\text{sid}_2, \Delta'_A)$  to get  $[\Delta'_A]_{\beta_0}$  and  $[\Delta'_A]_{\Delta_B}$ , and then locally convert to  $[\beta_0]_{\Delta'_A}$  and  $[\Delta_B]_{\Delta'_A}$ . Then, both parties set  $\langle 1_B^{(2)} \rangle := \text{Convert1}_{[1] \rightarrow \langle \cdot \rangle}([\Delta_B]_{\Delta'_A})$ .
6. For  $w \in \mathcal{I}_A$ ,  $P_A$  and  $P_B$  set  $[b_w] = [0]$ ; for  $w \in \mathcal{I}_B$ , both parties set  $[a_w] = [0]$ . For each wire  $w \in \mathcal{I}_A \cup \mathcal{W}$ , two parties define  $[a_w]$  in  $[a]$  as the authenticated bit on wire  $w$ ; for each wire  $w \in \mathcal{I}_B \cup \mathcal{W}$ , define  $[b_w]$  in  $[b]$  as the authenticated bit on wire  $w$ . In a topological order, for each gate  $(i, j, k, T)$ ,  $P_A$  and  $P_B$  do the following:
  - If  $T = \oplus$ , compute  $[a_k] := [a_i] \oplus [a_j]$  and  $[b_k] := [b_i] \oplus [b_j]$ .
  - If  $T = \wedge$ ,  $P_A$  computes  $a_{i,j} := a_i \wedge a_j$ , and  $P_B$  computes  $b_{i,j} := b_i \wedge b_j$ .
7. Both parties run  $\text{Fix}(\text{sid}_0, \{b_{i,j}\}_{(i,j,*,\wedge) \in \mathcal{C}_{\text{and}}})$  to generate a set of authenticated bits  $\{[b_{i,j}]\}$ , and also execute  $\text{Fix}(\text{sid}_2, \{a_{i,j}\}_{(i,j,*,\wedge) \in \mathcal{C}_{\text{and}}})$  to generate a set of authenticated bits  $\{[a_{i,j}]\}$ .
8. For  $i \in [1, n], j \in [1, L]$ ,  $P_A$  and  $P_B$  set  $\langle a_i b_j^* \rangle := \text{Convert2}_{[1] \rightarrow \langle \cdot \rangle}([a_i]_{\beta_j}, \langle b_j^* \rangle)$ . Then, both parties collect these dual-key authenticated bits to obtain  $\langle a_i b^* \rangle$ , and compute  $\langle a_i b_j \rangle$  and  $\langle a_j b_i \rangle$  for each AND gate  $(i, j, k, \wedge)$  from  $\mathbf{M} \cdot \langle a_i b^* \rangle$  for  $i \in [1, n]$ . Further, both parties set  $\langle \hat{a}_k \rangle := \text{Convert2}_{[1] \rightarrow \langle \cdot \rangle}([\hat{a}_k]_{\beta_0}, \langle 1 \rangle)$  and  $\langle a_{i,j} \rangle \leftarrow \text{Convert2}_{[1] \rightarrow \langle \cdot \rangle}([a_{i,j}]_{\beta_0}, \langle 1 \rangle)$ .

**Fig. 5.** The compressed preprocessing protocol for a Boolean circuit  $\mathcal{C}$ .

*Communication Complexity.* As recent PCG-like COT protocols have communication complexity sublinear to the number of resulting correlations, we can ignore the communication cost of generating random COT correlations when

**Protocol  $\Pi_{\text{cpre}}$ , continued**

9. For each AND gate  $(i, j, k, \wedge)$ ,  $P_A$  and  $P_B$  locally compute  $\langle \tilde{b}_k \rangle := \langle a_{i,j} \rangle \oplus \langle a_i b_j \rangle \oplus \langle a_j b_i \rangle \oplus \langle \hat{a}_k \rangle$ . Then, for each  $k \in \mathcal{W}$ ,  $P_A$  sends  $\text{lsb}(\text{D}_A[\tilde{b}_k])$  to  $P_B$ , who computes  $b_k := \text{lsb}(\text{D}_A[\tilde{b}_k]) \oplus \text{lsb}(\text{D}_B[\tilde{b}_k])$ . For each AND gate  $(i, j, k, \wedge)$ ,  $P_B$  computes  $\hat{b}_k := \tilde{b}_k \oplus b_{i,j}$ .
10. Both parties run  $\text{Fix}(\text{sid}_0, \{\hat{b}_k\}_{k \in \mathcal{W}})$  to obtain  $[\hat{b}_k]$  for each  $k \in \mathcal{W}$ .

**Consistency check:**  $P_A$  and  $P_B$  perform the following consistency-check steps:

11. Let  $[B_i^*] = [b_i^* \Delta_B]_{\Delta_A}$  produced in the previous phase. Both parties call  $\mathcal{F}_{\text{DVKZ}}$  to prove the following statements hold:
  - For each AND gate  $(i, j, k, \wedge)$ , for  $([b_i], [b_j], [b_{i,j}])$ ,  $b_{i,j} = b_i \wedge b_j$ .
  - For each AND gate  $(i, j, k, \wedge)$ , for  $([a_i], [a_j], [a_{i,j}])$ ,  $a_{i,j} = a_i \wedge a_j$ .
  - For each  $i \in [1, L]$ , for  $([b_i^*], [\Delta_B], [B_i^*])$ ,  $B_i^* = b_i^* \cdot \Delta_B$ .
12.  $P_A$  and  $P_B$  call  $\mathcal{F}_{\text{COT}}$  on respective input  $(\text{init}, \text{sid}_3, \Delta'_A)$  and  $(\text{init}, \text{sid}_3)$ . Then they run  $[\Delta_B]_{\Delta'_A} := \text{Fix}(\text{sid}_3, \Delta_B)$  and  $\langle 1_B^{(3)} \rangle := \text{Convert1}_{[\cdot] \rightarrow \langle \cdot \rangle}([\Delta_B]_{\Delta'_A})$ .  $P_A$  and  $P_B$  run  $\text{CheckZero2}(\langle 1_B^{(1)} \rangle - \langle 1_B^{(2)} \rangle, \langle 1_B^{(2)} \rangle - \langle 1_B^{(3)} \rangle)$  and  $\text{EQCheck}([\Delta_B]_{\Delta_A}, [\Delta_B]_{\Delta'_A})$  to check that  $\Delta'_A, \Delta_B$  are consistent when it is used in different functionalities. Both parties run  $[\beta_i]_{\Delta_A^{-1}} \leftarrow \text{Invert}([b_i^* \Delta_B]_{\Delta_A})$  for each  $i \in [0, L]$ , and then execute  $\text{EQCheck}(\{[\beta_i]_{\Delta_A^{-1}}\}_{i \in [0, L]}, \{[\beta_i]_{\Delta'_A}\}_{i \in [0, L]})$ .
13.  $P_A$  and  $P_B$  call  $\mathcal{F}_{\text{COT}}$  on input  $(\text{extend}, \text{sid}_0, \kappa)$  to generate a vector of random authenticated bits  $[r]_B$  with  $r \in \mathbb{F}_2^\kappa$ , and run  $[r]_B \leftarrow \text{B2F}([r]_B)$  where  $r = \sum_{i \in [0, \kappa]} r_i \cdot X^i \in \mathbb{F}_{2^\kappa}$ . Then both parties run  $\text{Fix}(\text{sid}_0, r \cdot \Delta_B)$  to obtain  $[r \cdot \Delta_B]_{\Delta_A}$ . The parties execute  $\langle r \rangle \leftarrow \text{Convert1}_{[\cdot] \rightarrow \langle \cdot \rangle}([r \cdot \Delta_B]_{\Delta_A})$ .
14.  $P_A$  and  $P_B$  call  $\mathcal{F}_{\text{Rand}}$  to sample a random element  $\chi \in \mathbb{F}_{2^\kappa}$ .
15.  $P_A$  convinces  $P_B$  that  $\tilde{b}_k$  is correct (and thus  $\hat{b}_k$  is correct) for  $k \in \mathcal{W}$  as follows.
  - (a) Both parties compute  $\langle y \rangle := \sum_{k \in \mathcal{W}} \chi^k \cdot \langle \tilde{b}_k \rangle + \langle r \rangle$ . Then  $P_B$  sends  $y$  to  $P_A$ .
  - (b) The parties execute  $\text{CheckZero2}(\langle y \rangle - y \cdot \langle 1 \rangle)$ .
16.  $P_B$  convinces  $P_A$  that  $[\hat{b}_k]$  is correct for  $k \in \mathcal{W}$  as follows:
  - (a) For each AND gate  $(i, j, k, \wedge)$ ,  $P_A$  and  $P_B$  compute  $[\tilde{b}_k]_B := [\hat{b}_k]_B \oplus [b_{i,j}]_B$ .
  - (b) Both parties compute  $[y]_B := \sum_{k \in \mathcal{W}} \chi^k \cdot [\tilde{b}_k]_B + [r]_B$ .
  - (c)  $P_A$  and  $P_B$  run  $\text{CheckZero}([y]_B - y)$ .

**Output:**  $P_A$  and  $P_B$  output a matrix  $\mathbf{M}$  along with  $([a], [\hat{a}], [b^*], [\hat{b}])$ .

**Fig. 6.** The compressed preprocessing protocol for a Boolean circuit  $\mathcal{C}$ , continued.

counting the communication amortized to every triple. Our checking protocols only introduce a negligibly small communication overhead. Therefore, the  $\text{Fix}$  procedure brings the main communication cost where  $\text{Fix}$  is used to transform random COT to chosen COT. Also, since parameter  $L$  is logarithmic to the number  $n$  of triples, we only need to consider the  $\text{Fix}$  procedures related to  $n$ .

This includes IT-MAC generation of  $a_{i,j}$  (from  $P_A$  to  $P_B$  in step 6 of Fig. 5),  $b_{i,j}$  (from  $P_B$  to  $P_A$  in the same step),  $\hat{b}_k$  (from  $P_B$  to  $P_A$  in step 10 of Fig. 6). In addition, for each triple,  $P_A$  needs to send  $\text{lsb}(\text{D}[\tilde{b}_k])$  to  $P_B$  in step 9 of Fig. 6. Overall, the one-way communication cost is 2 bits per triple.

## 5 Authenticated Garbling from COT

Now we describe the online phase of our two-party computation protocol. We first introduce a generalized distributed garbling syntax which can be instantiated by different schemes and then introduce the complete Boolean circuit evaluation protocol  $\Pi_{2PC}$ .

### 5.1 Distributed Garbling

We define the format of distributed garbling using two macros `Garble` and `Eval`, assuming that the preprocessing information is ready. Notice that these two macros can be instantiated by different garbling schemes. In our main protocol that optimizes towards one-way communication we instantiate it using the distributed half-gates garbling [29] whereas we use the optimized WRK garbling of Dittmer et al. [16] for the version that optimizes towards two-way communication. We recall the respective schemes in the full version [14].

- `Garble(C)`:  $P_A$  and  $P_B$  perform *local* operations as follows:
  - $P_A$  computes and outputs  $(\mathcal{GC}_A, \{\mathbb{L}_{w,0}, \mathbb{L}_{w,1}\}_{w \in \mathcal{I}_A \cup \mathcal{I}_B \cup \mathcal{W} \cup \mathcal{O}})$ .
  - $P_B$  computes and outputs  $\mathcal{GC}_B$ .
- `Eval( $\mathcal{GC}_A, \mathcal{GC}_B, \{(A_w, \mathbb{L}_{w,A_w})\}_{w \in \mathcal{I}_A \cup \mathcal{I}_B}$ )`:  $P_B$  evaluates the garbled circuit and obtain  $\{A_w, \mathbb{L}_{w,A_w}\}_{w \in \mathcal{W} \cup \mathcal{O}}$ .

The addition of evaluator’s random masks is to decouple the abort probability with the real input values (recall that the `Eval` function only requires masked values). The following definition captures this security property.

**Definition 2.** *For a distributed garbling scheme with preprocessing defined by `Garble` and `Eval`, consider the event `Bad` where the evaluator aborts or outputs masked wire value  $A_w$  that is incorrect (wrt. the input values of `Eval` and the masks of preprocessing). We call a distributed garbling scheme to be  $\epsilon$ -selective failure resilience, if conditioned on the garbled circuit  $\mathcal{GC}_A, \mathcal{GC}_B$ , the evaluator’s candidate input wire labels  $\{\mathbb{L}_{w,0}, \mathbb{L}_{w,1}\}_{w \in \mathcal{I}_B}$  and the garbler’s input wire masked values and labels  $\{(A_w, \mathbb{L}_w)\}_{w \in \mathcal{I}_A}$ , for any two pairs of  $P_B$ ’s inputs  $\mathbf{y}, \mathbf{y}'$ , we have*

$$|\Pr[\text{Bad}|\mathbf{y}] - \Pr[\text{Bad}|\mathbf{y}']| \leq \epsilon ,$$

where  $\Pr[\text{Bad}|\mathbf{y}]$  denotes the probability that the event `Bad` happens when the evaluator’s input value is  $\mathbf{y}$  and with aforementioned conditions.

With uncompressed preprocessing the DILO-WRK and KRRW distributed garbling (recalled in the full version [14].) has 0-selective failure resilience [29, 36] since the inputs  $A_w$  to `Eval` are completely masked and independent of the real input. In Lemma 9 we show that for the DILO-WRK and KRRW schemes, replacing the evaluator’s mask to  $\rho$ -wise independent randomness induces  $2^{-\rho}$ -selective failure resilience.

The next lemma states that after evaluating the garbled circuit the garbler and evaluator implicitly holds the authentication of the masked public wire values (color/permutation bits). To the best of our knowledge we are the first to apply this observation in the consistency check of authenticated garbling.

**Lemma 7.** *After running Eval, the evaluator holds the ‘color bits’  $\Lambda_w$  for every wire  $w \in \mathcal{W}$ . The garbler  $P_A$  and evaluator  $P_B$  also hold  $K_A[\Lambda_w], M_B[\Lambda_w]$  subject to  $M_B[\Lambda_w] = K_A[\Lambda_w] + \Lambda_w \Delta_A$ .*

*Proof.* We can define the following values using only wire labels:

$$\Lambda_w := (\mathbf{L}_{w,0} \oplus \mathbf{L}_{w,\Lambda_w}) \cdot \Delta_A^{-1}, \quad M_B[\Lambda_w] := \mathbf{L}_{w,\Lambda_w}, \quad K_A[\Lambda_w] := \mathbf{L}_{w,0} .$$

It is easy to verify  $M_B[\Lambda_w] = K_A[\Lambda_w] + \Lambda_w \cdot \Delta_A$ , which implies that  $[\Lambda_w]_B := (\mathbf{L}_{w,0}, \mathbf{L}_{w,\Lambda_w}, \Lambda_w)$  is a valid IT-MAC.

## 5.2 A Dual Execution Protocol Without Leakage

We describe a malicious secure 2PC protocol with almost the same one-way communication as half-gates garbling. We achieve this by adapting the dual execution technique to the distributed garbling setting. Intuitively, our observation in Lemma 7 allows us to check the consistency of every wire of the circuit. Together with some IT-MAC techniques to ensure input consistency, our protocol circumvents the one-bit leakage of previous dual execution protocols [27, 28].

In the following descriptions, we denote the actual value induced by the input on each wire  $w$  of the circuit  $\mathcal{C}$  by  $z_w$ . The masked value on that wire is denoted as  $\Lambda_w := z_w \oplus a_w \oplus b_w$  which is revealed to the evaluator during evaluation. The protocol is described in Fig. 7 and Fig. 8.

*Intuitions of Consistency Checking.* The security of the semi-honest garbled circuit guarantees that when the garbled circuit is correctly computed, then except with negligible probability the evaluator can only acquire one of the two labels (corresponding to the execution path) for each wire in the circuit. Thus, we can check the color bits of the honest party against the labels that the corrupted party acquires (in the separate execution) to verify consistency.

Using the notations from Lemma 7, let  $\bar{\Lambda}_w := (\mathbf{L}_{w,\Lambda_w} \oplus \mathbf{L}_{w,0}) \cdot \Delta_A^{-1}, \bar{\Lambda}'_w := (\mathbf{L}'_{w,\Lambda'_w} \oplus \mathbf{L}'_{w,0}) \cdot \Delta_B^{-1}$  for  $w \in \mathcal{W}$ . Our goal is to check the following equations where the left-hand (resp. right-hand) side is the evaluation result of  $P_A$  (resp.  $P_B$ ).

$$\bar{\Lambda}'_w \oplus a'_w \oplus b'_w = \Lambda_w \oplus a_w \oplus b_w \text{ for the corrupted } P_A \text{ case,} \quad (1)$$

$$\Lambda'_w \oplus a'_w \oplus b'_w = \bar{\Lambda}_w \oplus a_w \oplus b_w \text{ for the corrupted } P_B \text{ case.} \quad (2)$$

Multiplying the first equation by  $\Delta_B$ , the second by  $\Delta_A$  and do summation<sup>3</sup> gives the  $\tilde{V}_w^A, \tilde{V}_w^B$  values in the consistency checking.

$$\begin{aligned} (a_w + a'_w + \Lambda'_w) \Delta_A + M_A[a_w + a'_w] &= (b_w + b'_w + \Lambda_w) \Delta_B + M_B[b_w + b'_w] \\ + M_A[\Lambda'_w] + K_A[b_w + b'_w + \Lambda_w] &= + M_B[\Lambda_w] + K_B[a_w + a'_w + \Lambda'_w] \end{aligned}$$

<sup>3</sup> We define  $a_w, a'_w, b_w, b'_w$  by the MAC tag and keys to implicitly authenticate them.

**Protocol  $\Pi_{2PC}$** 

**Inputs:** In the preprocessing phase,  $P_A$  and  $P_B$  agree on a Boolean circuit  $\mathcal{C}$  with circuit-input wires  $\mathcal{I}_A \cup \mathcal{I}_B$ , output wires of all AND gates  $\mathcal{W}$  and circuit-output wires  $\mathcal{O}$ . In the online phase,  $P_A$  holds an input  $x \in \{0, 1\}^{|\mathcal{I}_A|}$  and  $P_B$  holds an input  $y \in \{0, 1\}^{|\mathcal{I}_B|}$ ;  $P_B$  will receive the output  $z = \mathcal{C}(x, y)$ . Let  $H : \{0, 1\}^{2\kappa} \rightarrow \{0, 1\}^\kappa$  and  $H' : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa$  be two random oracles.

**Preprocessing:**  $P_A$  plays the role of a garbler and  $P_B$  acts as an evaluator, and two parties execute as follows:

1. Both parties call  $\mathcal{F}_{\text{cpre}}$  to obtain a matrix  $\mathbf{M}$  and vectors of authenticated bits  $([\mathbf{a}], [\hat{\mathbf{a}}], [\mathbf{b}^*], [\hat{\mathbf{b}}])$ . The parties locally compute  $[\mathbf{b}] := \mathbf{M} \cdot [\mathbf{b}^*]$ .
2. Following a predetermined topological order,  $P_A$  and  $P_B$  use  $([\mathbf{a}], [\hat{\mathbf{a}}], [\mathbf{b}], [\hat{\mathbf{b}}])$  to obtain authenticated masks  $[a_w], [b_w]$  for each wire  $w$  and other authenticated bits that will be used in the construction of authenticated garbling.
3. Using the authenticated bits from the previous step and the KRRW garbling scheme,  $P_A$  and  $P_B$  run **Garble** to generate a distributed garbled circuit  $(\mathcal{G}\mathcal{C}_A, \mathcal{G}\mathcal{C}_B)$ , and  $P_A$  sends  $\mathcal{G}\mathcal{C}_A$  to  $P_B$ . For each wire  $w$ , two garbled labels  $L_{w,0}, L_{w,1} \in \{0, 1\}^\kappa$  are generated and satisfy  $L_{w,1} = L_{w,0} \oplus \Delta_A$ .  $P_A$  knows the label  $L_{w,0}$  for each wire  $w$  as well as  $\Delta_A$ .

**Online:** In the following steps,  $P_A$  securely transmits one label on each circuit-input wire to  $P_B$ , and  $P_B$  evaluates the circuit.

4. For each  $w \in \mathcal{I}_A$ ,  $P_A$  computes a masked value  $A_w := x_w \oplus a_w \in \{0, 1\}$ , and then sends  $(A_w, L_{w,A_w})$  to  $P_B$ .
5.  $P_A$  and  $P_B$  call  $\mathcal{F}_{\text{COT}}$  on respective input  $(\text{init}, \text{sid}, \Delta_A)$  and  $(\text{init}, \text{sid})$ , and then send  $(\text{extend}, \text{sid}, |\mathcal{I}_B|)$  to  $\mathcal{F}_{\text{COT}}$ , which returns random authenticated bits  $[r]_B$  to the parties.
6. For each  $w \in \mathcal{I}_B$ ,  $P_B$  computes  $A_w := y_w \oplus b_w$  and then sends  $d_w := A_w \oplus r_w$  to  $P_A$ . Both parties set  $[A_w]_B := [r_w]_B \oplus d_w$ . For each  $w \in \mathcal{I}_B$ ,  $P_A$  sends  $m_{w,0} := H(K_A[A_w], w||1) \oplus L_{w,0}$  and  $m_{w,1} := H(K_A[A_w] \oplus \Delta_A, w||1) \oplus L_{w,1}$  to  $P_B$ , who computes  $L_{w,A_w} := m_{w,A_w} \oplus H(M_B[A_w], w||1)$ .
7.  $P_B$  runs **Eval** $(\mathcal{G}\mathcal{C}_A, \mathcal{G}\mathcal{C}_B, \{(A_w, L_{w,A_w})\}_{w \in \mathcal{I}_A \cup \mathcal{I}_B})$  to obtain  $(A_w, L_{w,A_w})$  for each wire  $w \in \mathcal{W} \cup \mathcal{O}$ . For each  $w \in \mathcal{W}$ , both parties define  $[A_w]_B = (L_{w,0}, L_{w,A_w}, A_w)$ .

**Fig. 7.** Actively secure 2PC protocol in the  $\mathcal{F}_{\text{cpre}}$ -hybrid model.

*Communication Complexity.* In our dual execution protocol,  $P_A$  and  $P_B$  sends  $(2\kappa + 1)t + (\kappa + 1)|\mathcal{I}_A| + 2\kappa|\mathcal{I}_B| + \kappa + |\mathcal{O}|$  and  $(2\kappa + 1)t + (\kappa + 2)|\mathcal{I}_B| + 2\kappa|\mathcal{I}_A|$  bits respectively. Therefore the amortized one-way communication is  $2\kappa + 1$  bits per AND gate. Since we need to call  $\mathcal{F}_{\text{cpre}}$  twice in  $\Pi_{2PC}$ , we conclude that the amortized one-way (resp. two-way) communication in the  $(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\text{bcOT}}, \mathcal{F}_{\text{DVZK}}, \mathcal{F}_{\text{EQ}}, \mathcal{F}_{\text{Rand}})$ -hybrid model is  $2\kappa + 5$  (resp.  $4\kappa + 10$ ) bits.

For the second version that combines  $\Pi_{\text{cpre}}$  and the optimized WRK online protocol, the amortized one-way (resp. two-way) communication is  $2\kappa + 3\rho + 2$  (resp.  $2\kappa + 3\rho + 4$ ) bits in the same hybrid model.

Protocol  $\Pi_{2PC}$ , continued**Dual execution and consistency check:**

8. Re-using the initialization procedure of functionality  $\mathcal{F}_{\text{cpre}}$  (i.e., the same global keys  $\Delta_A$  and  $\Delta_B$  are adopted),  $P_A$  and  $P_B$  execute the preprocessing phase as described above again by swapping the roles (i.e.,  $P_A$  is an evaluator and  $P_B$  is a garbler). Thus, for each  $w \in \mathcal{W}$ ,  $P_A$  and  $P_B$  hold  $[a'_w]$  and  $[b'_w]$ . For each wire  $w$ ,  $P_B$  has also the label  $L'_{w,0}$ .
9. Swapping the roles (i.e.,  $P_A$  is the evaluator and  $P_B$  is the garbler),  $P_A$  and  $P_B$  execute the online phase as described above again, except for the following differences of processing inputs:
  - (a) For each  $w \in \mathcal{I}_B$ ,  $P_A$  and  $P_B$  run  $\text{Open}([b_w] \oplus [b'_w] \oplus [r_w]_B \oplus d_w)$  that enables  $P_A$  to obtain the masked value  $\Lambda'_w = y_w \oplus b'_w$ , and  $P_B$  sends  $L'_{w,\Lambda'_w}$  to  $P_A$ .
  - (b) For each  $w \in \mathcal{I}_A$ , both parties set  $[\Lambda'_w]_A := [a_w] \oplus [a'_w] \oplus \Lambda_w$ , and then garbler  $P_B$  sends  $m'_{w,0} := \text{H}(\text{K}_B[\Lambda'_w], w \| 2) \oplus L'_{w,0}$  and  $m'_{w,1} := \text{H}(\text{K}_B[\Lambda'_w] \oplus \Delta_B, w \| 2) \oplus L'_{w,1}$  to  $P_A$ , who computes  $L'_{w,\Lambda'_w} := m'_{w,\Lambda'_w} \oplus \text{H}(\text{M}_A[\Lambda'_w], w \| 2)$ .
 After the 2th execution of online phase,  $P_A$  and  $P_B$  obtain  $[\Lambda'_w]_A$  for all  $w \in \mathcal{W}$ .
10.  $P_A$  and  $P_B$  check that  $(\Lambda_w \oplus a_w \oplus b_w) \cdot (\Delta_A \oplus \Delta_B) = (\Lambda'_w \oplus a'_w \oplus b'_w) \cdot (\Delta_A \oplus \Delta_B)$  holds by performing the following steps.
  - (a) For each  $w \in \mathcal{W}$ ,  $P_A$  and  $P_B$  respectively compute
 
$$V_w^A = (a_w \oplus a'_w \oplus \Lambda'_w) \Delta_A \oplus \text{M}_A[a_w] \oplus \text{M}_A[a'_w] \oplus \text{M}_A[\Lambda'_w] \oplus \text{K}_A[b_w] \\ \oplus \text{K}_A[b'_w] \oplus \text{K}_A[\Lambda_w], \text{B} = (b_w \oplus b'_w \oplus \Lambda_w) \Delta_B \\ \oplus \text{M}_B[b_w] \oplus \text{M}_B[b'_w] \oplus \text{M}_B[\Lambda_w] \oplus \text{K}_B[a_w] \oplus \text{K}_B[a'_w] \oplus \text{K}_B[\Lambda'_w].$$
  - (b)  $P_A$  computes  $h := \text{H}'(V_1^A, \dots, V_t^A)$ , and then sends it to  $P_B$  who checks that  $h = \text{H}'(V_1^B, \dots, V_t^B)$ . If the check fails,  $P_B$  aborts.

**Output processing:** For each  $w \in \mathcal{O}$ ,  $P_A$  and  $P_B$  run  $\text{Open}([a_w])$  such that  $P_B$  receives  $a_w$ , and then  $P_B$  computes  $z_w := \Lambda_w \oplus (a_w \oplus b_w)$ .

**Fig. 8.** Actively secure 2PC protocol in the  $\mathcal{F}_{\text{cpre}}$ -hybrid model, continued.

### 5.3 Security Analysis

We first give two useful lemmas about the equality checking (following the proofs of [17, 29, 36]) refer to the full version [14] for their proofs. We state the security of our 2PC protocol in Theorem 2 and prove it in the full version [14].

**Lemma 8.** *After the equality check, except with probability  $\frac{2+\text{poly}(\kappa)}{2^\kappa}$ ,  $P_B$  either aborts or evaluates the garbled circuit exactly according to  $\mathcal{C}(\mathbf{x}, \mathbf{y})$ , where we canonically define the circuit input  $\mathbf{x}, \mathbf{y}$  using the messages in step 4, step 6, and the randomness from the preprocessing phase.*

**Lemma 9.** *For the DILO-WRK and KRRW distributed garbling schemes (see details in the full version [14].) by sampling the wire masks  $\mathbf{a}, \mathbf{a}', \mathbf{b}, \mathbf{b}'$  using the compressed preprocessing functionality  $\mathcal{F}_{\text{cpre}}$  (recall that  $\mathbf{b} := \mathbf{M} \cdot \mathbf{b}^*$ ,  $\mathbf{a}' := \mathbf{M} \cdot (\mathbf{a}^*)'$  are compressed randomness), the resulting schemes have  $2^{-\rho}$ -selective failure resilience.*

**Theorem 2.** *Protocol  $\Pi_{2PC}$  shown in Fig. 7 and Fig. 8 securely realizes functionality  $\mathcal{F}_{2PC}$  in the presence of malicious adversary in the  $\mathcal{F}_{\text{cpre}}$ -hybrid model and the random oracle model.*

**Acknowledgements.** Kang Yang is supported by the National Key Research and Development Program of China (Grant No. 2022YFB2702000), and by the National Natural Science Foundation of China (Grant Nos. 62102037, 61932019, 62022018). Yu Yu is supported by the National Natural Science Foundation of China (Grant Nos. 62125204 and 92270201), the National Key Research and Development Program of China (Grant No. 2018YFA0704701), and the Major Program of Guangdong Basic and Applied Research (Grant No. 2019B030302008). Yu Yu also acknowledges the support from the XPLOER PRIZE. Xiao Wang is supported by DARPA under Contract No. HR001120C0087, NSF award #2016240, #2236819, and research awards from Meta and Google. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. We thank anonymous reviewers for their helpful comments.

## References

1. Abascal, J., Sereshgi, M.H.F., Hazay, C., Ishai, Y., Venkatasubramanian, M.: Is the classical GMW paradigm practical? the case of non-interactive actively secure 2PC. In: ACM Conference on Computer and Communications Security (CCS) 2020, pp. 1591–1605. ACM Press (2020). <https://doi.org/10.1145/3372297.3423366>
2. Baum, C., Braun, L., Munch-Hansen, A., Razet, B., Scholl, P.: Appenzeller to bribe: efficient zero-knowledge proofs for mixed-mode arithmetic and  $\mathbb{Z}_{2^k}$ . In: ACM Conference on Computer and Communications Security (CCS) 2021, pp. 192–211. ACM Press (2021). <https://doi.org/10.1145/3460120.3484812>
3. Baum, C., Braun, L., Munch-Hansen, A., Scholl, P.: Moz $\mathbb{Z}_{2^k}$  arella: efficient vector-OLE and zero-knowledge proofs over  $\mathbb{Z}_{2^k}$ . In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 329–358. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-15985-5\\_12](https://doi.org/10.1007/978-3-031-15985-5_12)
4. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac’n’Cheese: zero-knowledge proofs for Boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 92–122. Springer, Cham (2021)
5. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd Annual ACM Symposium on Theory of Computing (STOC), pp. 503–513. ACM Press (1990). <https://doi.org/10.1145/100216.100287>
6. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: IEEE Symposium on Security and Privacy (S&P) 2013, pp. 478–492 (2013). <https://doi.org/10.1109/SP.2013.39>
7. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-20465-4\\_11](https://doi.org/10.1007/978-3-642-20465-4_11)
8. Blum, A., Furst, M., Kearns, M., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 278–291. Springer, Heidelberg (1994). [https://doi.org/10.1007/3-540-48329-2\\_24](https://doi.org/10.1007/3-540-48329-2_24)



9. Boyle, E., et al.: Correlated pseudorandomness from expand-accumulate codes. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 603–633. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-15979-4\\_21](https://doi.org/10.1007/978-3-031-15979-4_21)
10. Boyle, E., et al.: Efficient two-round OT extension and silent non-interactive secure computation. In: ACM Conference on Computer and Communications Security (CCS) 2019, pp. 291–308. ACM Press (2019). <https://doi.org/10.1145/3319535.3354255>
11. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 489–518. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-26954-8\\_16](https://doi.org/10.1007/978-3-030-26954-8_16)
12. Canetti, R.: Security and composition of multiparty cryptographic protocols. *J. Cryptology* **13**(1), 143–202 (2000). <https://doi.org/10.1007/s001459910006>
13. Couteau, G., Rindal, P., Raghuraman, S.: Silver: silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 502–534. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84252-9\\_17](https://doi.org/10.1007/978-3-030-84252-9_17)
14. Cui, H., Wang, X., Yang, K., Yu, Y.: Actively Secure Half-Gates with Minimum Overhead under Duplex Networks. *Cryptology ePrint Archive*, Paper 2023/278 (2023). <https://eprint.iacr.org/2023/278>
15. Damgård, I., Nielsen, J.B., Nielsen, M., Ranellucci, S.: The TinyTable protocol for 2-party secure computation, or: gate-scrambling revisited. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 167–187. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63688-7\\_6](https://doi.org/10.1007/978-3-319-63688-7_6)
16. Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Authenticated garbling from simple correlations. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 57–87. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-15985-5\\_3](https://doi.org/10.1007/978-3-031-15985-5_3)
17. Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Improving line-point zero knowledge: two multiplications for the price of one. In: ACM Conference on Computer and Communications Security (CCS) 2022, pp. 829–841. ACM Press (2022). <https://doi.org/10.1145/3548606.3559385>
18. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-point zero knowledge and its applications. In: 2nd Conference on Information-Theoretic Cryptography (2021)
19. 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](https://doi.org/10.1007/3-540-47721-7_12)
20. Goldreich, O.: *Foundations of Cryptography: Basic Applications*, vol. 2. Cambridge University Press, Cambridge, UK (2004)
21. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: 19th Annual ACM Symposium on Theory of Computing (STOC), pp. 218–229. ACM Press (1987). <https://doi.org/10.1145/28395.28420>
22. Guo, C., Katz, J., Wang, X., Weng, C., Yu, Y.: Better Concrete security for half-gates garbling (in the multi-instance setting). In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 793–822. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-56880-1\\_28](https://doi.org/10.1007/978-3-030-56880-1_28)
23. Guo, C., Katz, J., Wang, X., Yu, Y.: Efficient and secure multiparty computation from fixed-key block ciphers. In: IEEE Symposium on Security and Privacy (S&P) 2020, pp. 825–841 (2020). <https://doi.org/10.1109/SP40000.2020.00016>

24. Hazay, C., Ishai, Y., Venkatasubramanian, M.: Actively secure garbled circuits with constant communication overhead in the plain model. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 3–39. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-70503-3\\_1](https://doi.org/10.1007/978-3-319-70503-3_1)
25. Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round MPC combining BMR and oblivious transfer. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 598–628. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-70694-8\\_21](https://doi.org/10.1007/978-3-319-70694-8_21)
26. Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round MPC combining BMR and oblivious transfer. *J. Cryptology* **33**(4), 1732–1786 (2020). <https://doi.org/10.1007/s00145-020-09355-y>
27. Hazay, C., Shelat, A., Venkatasubramanian, M.: Going beyond dual execution: MPC for functions with efficient verification. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 328–356. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45388-6\\_12](https://doi.org/10.1007/978-3-030-45388-6_12)
28. Huang, Y., Katz, J., Evans, D.: Quid-Pro-Quo-tocols: strengthening semi-honest protocols with dual execution. In: IEEE Symposium on Security and Privacy (S&P) 2012, pp. 272–284 (2012). <https://doi.org/10.1109/SP.2012.43>
29. Katz, J., Ranellucci, S., Rosulek, M., Wang, X.: Optimizing authenticated garbling for faster secure two-party computation. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 365–391. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-96878-0\\_13](https://doi.org/10.1007/978-3-319-96878-0_13)
30. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-70583-3\\_40](https://doi.org/10.1007/978-3-540-70583-3_40)
31. Lindell, Y., Pinkas, B., Smart, N.P., Yanai, A.: Efficient constant round multi-party computation combining BMR and SPDZ. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 319–338. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48000-7\\_16](https://doi.org/10.1007/978-3-662-48000-7_16)
32. Lindell, Y., Smart, N.P., Soria-Vazquez, E.: More efficient constant-round multi-party computation from BMR and SHE. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9985, pp. 554–581. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53641-4\\_21](https://doi.org/10.1007/978-3-662-53641-4_21)
33. Mohassel, P., Franklin, M.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006). [https://doi.org/10.1007/11745853\\_30](https://doi.org/10.1007/11745853_30)
34. 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](https://doi.org/10.1007/978-3-642-32009-5_40)
35. Rosulek, M., Roy, L.: Three halves make a whole? beating the half-gates lower bound for garbled circuits. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 94–124. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84242-0\\_5](https://doi.org/10.1007/978-3-030-84242-0_5)
36. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: ACM Conference on Computer and Communications Security (CCS) 2017, pp. 21–37. ACM Press (2017). <https://doi.org/10.1145/3133956.3134053>

37. Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: ACM Conference on Computer and Communications Security (CCS) 2017, pp. 39–56. ACM Press (2017). <https://doi.org/10.1145/3133956.3133979>
38. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for Boolean and arithmetic circuits. In: IEEE Symposium on Security and Privacy (S&P) 2021, pp. 1074–1091 (2021). <https://doi.org/10.1109/SP40001.2021.00056>
39. Weng, C., Yang, K., Xie, X., Katz, J., Wang, X.: Mystique: efficient conversions for zero-knowledge proofs with applications to machine learning. In: USENIX Security Symposium 2021, pp. 501–518. USENIX Association (2021)
40. Weng, C., Yang, K., Yang, Z., Xie, X., Wang, X.: AntMan: interactive zero-knowledge proofs with sublinear communication. In: ACM Conference on Computer and Communications Security (CCS) 2022, pp. 2901–2914. ACM Press (2022). <https://doi.org/10.1145/3548606.3560667>
41. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: ACM Conference on Computer and Communications Security (CCS) 2021, pp. 2986–3001. ACM Press (2021). <https://doi.org/10.1145/3460120.3484556>
42. Yang, K., Wang, X., Zhang, J.: More efficient MPC from improved triple generation and authenticated garbling. In: ACM Conference on Computer and Communications Security (CCS) 2020, pp. 1627–1646. ACM Press (2020). <https://doi.org/10.1145/3372297.3417285>
43. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: fast extension for correlated OT with small communication. In: ACM Conference on Computer and Communications Security (CCS) 2020, pp. 1607–1626. ACM Press (2020). <https://doi.org/10.1145/3372297.3417276>
44. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science (FOCS), pp. 162–167. IEEE (1986). <https://doi.org/10.1109/SFCS.1986.25>
45. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46803-6\\_8](https://doi.org/10.1007/978-3-662-46803-6_8)



# Black-Box Reusable NISC with Random Oracles

Yuval Ishai<sup>1</sup>, Dakshita Khurana<sup>2</sup>, Amit Sahai<sup>3</sup>,  
and Akshayaram Srinivasan<sup>4</sup>(✉)

<sup>1</sup> Technion, Haifa, Israel

<sup>2</sup> UIUC, Champaign, USA

<sup>3</sup> UCLA, Los Angeles, US

<sup>4</sup> Tata Institute of Fundamental Research, Mumbai, India  
akshayaram@berkeley.edu

**Abstract.** We revisit the problem of *reusable* non-interactive secure computation (NISC). A standard NISC protocol for a sender-receiver functionality  $f$  enables the receiver to encrypt its input  $x$  such that any sender, on input  $y$ , can send back a message revealing only  $f(x, y)$ . Security should hold even when either party can be malicious. A *reusable* NISC protocol has the additional feature that the receiver's message can be safely reused for computing multiple outputs  $f(x, y_i)$ . Here security should hold even when a malicious sender can learn partial information about the honest receiver's outputs in each session.

We present the first reusable NISC protocol for general functions  $f$  that only makes a *black-box* use of any two-message oblivious transfer protocol, along with a random oracle. All previous reusable NISC protocols either made a non-black-box use of cryptographic primitives (Cachin et al. ICALP 2002) or alternatively required a stronger arithmetic variant of oblivious transfer and were restricted to  $f$  in  $\text{NC}^1$  or similar classes (Chase et al. Crypto 2019). Our result is obtained via a general compiler from standard NISC to reusable NISC that makes use of special type of honest-majority protocols for secure multiparty computation.

Finally, we extend the above main result to reusable *two-sided* NISC, in which two parties can encrypt their inputs in the first round and then reveal different functions of their inputs in multiple sessions. This extension either requires an additional (black-box) use of additively homomorphic commitment or alternatively requires the parties to maintain a state between sessions.

## 1 Introduction

Consider the following minimal setting for secure computation. There are two parties, a *sender* and a *receiver*, and two rounds of interaction. In the first round, the receiver encrypts its input  $x$  and sends the resulting message  $\pi_1$  to the sender. In the second round, the sender uses the message  $\pi_1$  and its input  $y$  to compute a message  $\pi_2$ . Based on  $\pi_2$  and its secret randomness, the receiver should compute

the output  $f(x, y)$ , for some predetermined  $f$ , but should not learn additional information about the sender’s input  $y$ .

When the parties are semi-honest, the problem is relatively easy to solve by using garbled circuits [Yao86], under the (minimal) assumption that a two-message oblivious transfer (OT) protocol exists. When security needs to hold against malicious parties, the problem becomes more challenging, and is referred to as *non-interactive secure computation* (NISC) [IKO+11].

NISC is a powerful general-purpose tool for computing on encrypted data. For instance, NISC enables users (acting as receivers) to safely post their encrypted sensitive data on the internet, such that any other user (acting as a sender) can perform a secure computation with them, say to determine whether their profiles match, by sending a single message. However, despite a significant amount of research, all of the existing solutions to NISC are unsatisfactory in either of the following ways:

- *Non-black-Box Use of Cryptography.* The most natural approach for protecting NISC protocols against malicious parties is by using non-interactive zero-knowledge (NIZK) proofs for enforcing honest behavior [CCKM00, HK07, ASH+20]. However, this NIZK-based approach is typically quite impractical, resulting in orders of magnitude of slowdown compared to the semi-honest baseline. A good explanation for this is the fact that such NISC protocols make a *non-black-box* use of the underlying cryptographic primitives, requiring their explicit representation rather than just making an oracle use of their input-output relation.
- *Limited Reusability.* Motivated by the inefficiency of non-black-box protocols, several works obtained practical NISC protocols that make a black-box use of cryptographic primitives, typically only a two-message OT protocol and a pseudorandom generator<sup>1</sup> [IKO+11, AMPR14, MR17, DILO22]. In fact, when cast in the “OT-hybrid” model, where the parties can make parallel calls to an ideal OT oracle, these protocols are secure against a *computationally unbounded* malicious sender. Furthermore, they can efficiently achieve full information-theoretic security for functions  $f$  in  $\text{NC}^1$  and similar classes. A subtle but important vulnerability of these protocols is that they are not fully *reusable*: If the same receiver message is used in multiple sessions to generate malformed sender messages, supposedly for computing  $f(x, y_1), f(x, y_2), \dots$ , then even a partial leakage of the receiver’s outputs can lead to a total break of security. For example, in the case of zero-knowledge proofs, if the sender can learn whether the receiver accepts several malformed proofs of true statements, it can make the receiver accept a false statement. This limitation was shown in [CDI+19] to be inherent for NISC in the OT-hybrid model with a computationally unbounded sender: There are explicit functions  $f$  for which no such NISC protocol can be reusable.
- *Restricted Functionality.* To circumvent the above impossibility, Chase et al. [CDI+19] (with subsequent efficiency improvement in [DIO21]) suggested

---

<sup>1</sup> Since a pseudorandom generator can be constructed from an OT protocol in a black-box way, OT alone suffices.

the use of an arithmetic variant of OT, called oblivious linear evaluation (OLE), instead of standard OT. Their main positive result is an information-theoretic reusable NISC protocol for *arithmetic branching programs* in the OLE-hybrid model, efficiently capturing  $f$  in  $\text{NC}^1$  and similar classes. Beyond the limitation on  $f$ , the reusable flavor of OLE required by the protocol of Chase et al. [CDI+19] is only known from the DCR assumption [Pai99] (or alternatively requires preprocessing) and is considerably more expensive to realize than OT. While it was shown in [CDI+19] how to bootstrap from branching program to circuits, this step requires a non-black-box use of a pseudorandom generator.

The above state of affairs suggests the following open question:

Is there a general-purpose *reusable* NISC protocol that only makes a *black-box* use of a two-message OT protocol?

*Shouldn't this be Impossible?* Recall that the impossibility result from [CDI+19] rules out protocols that make parallel calls to an ideal OT oracle and achieve reusable security against a computationally unbounded sender. Then how can we hope to achieve the goals above? Our key idea for bypassing this impossibility result is to make black-box use of the *next-message functions* and *receiver output function* of a two-message OT protocol. Note that this is different than making black-box use of an ideal OT functionality because it allows for “explaining” the message produced by one of the next-message functions of the OT protocol by revealing the inputs and randomness used to produce that message. Nevertheless, except for the fact that we have to settle for computational security against a malicious sender, this still allows our protocol to make use of any off-the-shelf two-message OT protocol when instantiating our approach.

## 1.1 Our Contribution

Our main result is an affirmative answer to the above question in the *random oracle model*. This gives the first reusable NISC protocol for general functions  $f$  that only makes a black-box use of cryptography. In fact, we show the following more general result.

**Theorem 1 (Reusable NISC from NISC, Informal).** *There is a reusable NISC protocol for  $f$  in the random oracle model that makes a black-box use of any (non-reusable) NISC protocol for a related  $f'$ .*

The theorem is proved via a compiler from standard NISC to reusable NISC that makes use of special type of honest-majority MPC protocols. Note that standard NISC can be constructed from any two-message malicious OT in a black-box way [IKO+11]. Since two-message malicious OT can be obtained in a black-box way from two-message semi-honest OT in the random oracle model [IKSS22a], we can base our protocol on semi-honest OT. While in this work we focus on feasibility and do not attempt to optimize concrete efficiency, an optimized variant

of our construction is likely to yield reusable NISC protocols with good concrete efficiency.

Finally, we extend the above main result to a reusable *two-sided* variant of NISC, in which two parties can encrypt their inputs  $(x, y)$  and then reveal different functions  $f_i$  of their inputs in multiple sessions.<sup>2</sup> This extension makes a black-box use of a “reusable commit-and-prove” primitive which requires the commitments to the secret input to be reusable across different sessions with the verifier. We show how to construct such a primitive by making a black-box use of an additively homomorphic commitment scheme. Alternatively, we can construct this primitive unconditionally in the random oracle model if the parties can maintain an updatable state between sessions.

**Theorem 2 (Reusable Two-Sided NISC from NISC, Informal).** *Assume black-box access to a (non-reusable) one-sided NISC protocol and a non-interactive reusable commit-and-prove protocol. Then, there exists a reusable (two-sided) NISC protocol in the random oracle model.*

## 2 Technical Overview

In this section, we give a high-level overview of the key ideas behind our construction of a black-box reusable NISC protocol in the random oracle model. Later, we explain the additional challenges in extending these ideas to the two-sided setting and discuss our approaches to overcome them.

*Reusable NISC Protocol.* Recall that a non-interactive secure computation (NISC) protocol for a two-party functionality  $f$  is a two-message protocol between a receiver and a sender that delivers the output of  $f$  to the receiver. A NISC protocol is said to be reusable if the message from the honest receiver is fixed once and for all and the adversarial sender can execute multiple sessions with the honest receiver. In each such session, the adversarial sender generates a new second round message in the protocol and can learn the output computed by the honest receiver.<sup>3</sup> It can then adaptively decide to continue with the next session or stop the execution. We require the view of the adversarial sender, together with the output of the honest receiver, to be simulatable in an ideal world where the parties only have access to a trusted functionality that implements  $f$ .

---

<sup>2</sup> In fact, our result applies to a more general notion of two-sided NISC that strictly generalizes both the above notion and standard (one-sided) NISC.

<sup>3</sup> In the actual definition, we consider a more general situation where the adversary can learn some partial information about the output, such as whether the receiver aborts. This makes reusable security nontrivial even for functionalities such as OLE, where the receiver’s output reveals its input. However, for the sake of this overview, we make the simplifying assumption that the entire receiver output is given to the adversary.

*Impossibility in the OT-Hybrid Model.* Before we explain our solution, let us first recall the intuition, already discussed in [IKO+11], for why reusable security is challenging for “natural” NISC protocols. Let’s consider an honest receiver who has generated the first round message by making several calls to the OT oracle by acting as the OT receiver. We concentrate on one such call where the receiver’s choice bit is  $b$ . A malicious sender who tries to break the security of the protocol can make a guess  $b'$  for this bit and give two sender messages  $(m_0, m_1)$  such that  $m_{b'}$  is correctly generated as per the protocol specification but  $m_{1-b'}$  is malformed. It provides these two messages as the sender input to the OT oracle. Now, if the guess  $b'$  was correct, the honest receiver does not notice this and continues to compute the output. On the other hand, if the guess was incorrect, then the receiver obtains the malformed sender’s message. For natural NISC protocols, this makes the receiver abort. Thus, depending on whether the receiver aborts or not, the sender learns the value of the receiver’s choice bit  $b$  in this OT execution. This is not a major problem in the single-use NISC setting, as there are standard ways to secret-share the receiver’s input so that the receiver’s abort event is uncorrelated with its actual input. However, this has a devastating effect in the case of reusable security. Specifically, for each one of the OT executions with the receiver, the sender can learn its choice bit one-at-a-time by mounting the above attack across different sessions. Once the sender does this, there is no hope of protecting the privacy of the receiver’s input. Chase et al. [CDI+19] extended this argument to arbitrary protocols, showing that information-theoretic reusable NISC in the OT-hybrid model is impossible. This applies even to simple functionalities, such as the OLE<sup>4</sup> functionality, for which efficient information-theoretic protocols in the OT-hybrid model exist in the non-reusable NISC setting.

*Main Goals.* Somewhat surprisingly, Chase et al. showed that this impossibility result can be circumvented if we replace OT-hybrid with the OLE-hybrid model. Specifically, they proved that even after many sessions with an honest receiver, a malicious sender cannot obtain any advantage over an ideal execution. Intuitively, unlike the case of OT, each receiver’s input to the OLE oracle can only be guessed with negligible probability. This allows the receiver to detect every cheating attempt of the sender with overwhelming probability, thereby preventing the sender from gaining significant information about the OLE inputs. Chase et al. built on this idea and gave a construction of a reusable NISC in the OLE-hybrid model. This positive result showed that if we implement the OLE functionality using a two-message OLE protocol with reusable receiver security<sup>5</sup> in either

<sup>4</sup> OLE is the arithmetic analogue of OT which takes in a field element  $x$  from the receiver, and two field elements  $(a, b)$  from the sender and outputs  $ax + b$  to the receiver.

<sup>5</sup> In reusable receiver security game, we fix the first round message from the honest receiver and the corrupted sender could generate multiple second round messages. We require the joint distribution of the view of the sender and the receiver’s output in each of the sender executions to be indistinguishable to an ideal world where the parties have access to the ideal OLE functionality.



the CRS/RO model, then we have a reusable NISC protocol with same kind of setup. Unfortunately, such a two-message OLE protocol [CDI+19] is only known from the DCR assumption [Pai99] and makes heavy use of expensive public-key cryptography. Furthermore, Chase et al.’s construction for computing circuits (in contrast to the information-theoretic construction for branching programs) made non-black-box use of a PRG. Given the above state of the art, the two main goals of our work are:

1. Explore new approaches to bypass the impossibility in the OT-hybrid model without resorting to the more expensive OLE primitive.
2. Obtain reusable NISC for *circuits* while only making a black-box use of cryptography.

*Our Approach: Making Black-Box Use of Two-Message OT.* The key approach we take to bypass this impossibility result is to settle for *computational security* against a malicious sender, while only making a black-box use of a *two-message OT protocol*. Before we explain the technical ideas in our construction, let us first explain how black-box use of a two-message OT is different from treating the OT functionality as an oracle (as it is done in the OT-hybrid model). In the OT-hybrid model, the receiver and the sender have access to an OT functionality. The OT functionality takes a choice bit  $b$  from the receiver and two messages  $(m_0, m_1)$  from the sender and provides  $m_b$  to the receiver. The only interface that this model provides is to receive the private inputs from the parties and give outputs. In particular, there is no way to “connect” the inputs that the parties provide to this oracle with the other components in the protocol. On the other hand, in the black-box two-message OT setting<sup>6</sup>, we model the oblivious transfer using the cryptographic algorithms that implement this functionality. Specifically, we model a two-message OT protocol as a tuple of algorithms  $(\text{OT}_1, \text{OT}_2, \text{out}_{\text{OT}})$ .  $\text{OT}_1$  is run by the receiver and takes the receiver’s choice bit  $b$  and outputs the first round message  $\text{otm}_1$ .  $\text{OT}_2$  is run by the sender and takes the receiver’s message  $\text{otm}_1$ , the sender’s private input  $(m_0, m_1)$  and outputs the second round message  $\text{otm}_2$ .  $\text{out}_{\text{OT}}$  is run by the receiver and takes  $\text{otm}_2$  and the receiver’s private random tape and outputs  $m_b$ . Note that the interface that is provided by these oracles is to take inputs and randomness from the parties and provide *the protocol messages* that they need to send to the other parties. We model these messages as handles and importantly, these handles can be “opened” to the other party. Specifically, the parties can send the input and randomness used in generating these handles to the other party which can then check if this handle was generated correctly by querying the oracles. In other words, one can treat these handles as commitments to the sender and the receiver inputs to the OT functionality. As a result, we can use these commitments as a “link” between the inputs provided by the parties to the OT oracle and the rest of the protocol. In particular, this opens up new avenues to prove that the messages given to

---

<sup>6</sup> We restrict ourselves to the case of a two-message OT protocol as this gives a two-message NISC protocol.

these handles are well-formed and hence, do not give rise to an input-dependent abort. Such a mechanism was impossible to achieve in the OT-hybrid model.

*Challenges.* Can we use this observation to upgrade any NISC protocol in the black-box OT model to have security in the reusable setting? Unfortunately, this does not seem to be the case and let us explain why. Almost all known black-box constructions of NISC use a two-message OT protocol to implement a mechanism called as the watchlists [IPS08]. Roughly, the watchlist mechanism is a sophisticated cut-and-choose technique that delivers the input and randomness used by one of the parties in a subset of the executions *privately* to the other party. Each party then checks if the other party behaved honestly in the set of watched executions and if any deviation is detected, the party aborts. If the set of watched executions are chosen *randomly* and *privately*, then this check ensures that a majority of the unwatched executions are emulated honestly. Once this is ensured, all these works have developed clever approaches to robustly combine the outputs from the rest of the executions to compute the output of the functionality. For this to succeed, it is important that watched executions are hidden from the corrupted party before it generates its protocol message. This is typically done by implementing some version of a  $k$ -out-of- $m$  OT functionality where one party choose a random subset of size  $k$  as part of its watchlist and the functionality delivers the input and randomness of the other party corresponding to each execution in this set. This  $k$ -out-of- $m$  OT functionality is implemented via a black-box access to a 1-out-of-2 OT. Specifically, the receiver chooses a random subset of size  $k$  and computes an encoding of this set. Each bit of the encoding is used as the choice bit in an execution of an 1-out-of-2 OT protocol. Regrettably, this technique makes these constructions to again suffer from the same problem as the one described earlier. In particular, we observe that the sender can mount a similar selective failure attack (as in the OT-hybrid model) to learn encoding of the random subset sampled by the receiver one bit at a time. Once the sender learns this encoding, it can easily break the privacy of the receiver’s input and cheat in all other executions that are not watched.

At a high-level what this attack shows is that we cannot hope to achieve reusable security by relying on any mechanism that hides a part of the receiver’s randomness via an 1-out-of-2 OT. All such mechanisms are bound to be broken in the reusable setting as a malicious sender can learn this secret randomness bit-by-bit. In other words, we need a technique where the randomness used in generating the set of watched executions to come solely from the sender’s side. This is a bit counter-intuitive as it seems to give the sender the power to fix this secret randomness to any value. Once the sender knows this value it can trivially cheat in the other unopened executions and break the security of the protocol.

*Random Oracles to the Rescue.* We overcome this conundrum by using random oracles to sample the set of watched executions. Specifically, we pass the sender’s message through a random oracle and this gives a subset of the executions to be opened. The correlation-intractability of the random oracle guarantees that the sender does not have the power to fix this set of opened executions to any

value of its choice. Importantly, we can ensure that this property holds even in the reusable setting as we can treat the output to every (new) query made to the random oracle as an independently chosen random subset. This idea of using random oracles to sample the set of watched executions is due to Ishai et al. [IKSS22a]. However, their motivation was to remove the use of malicious-secure OTs from the watchlist mechanism whereas our motivation is to obtain a construction in the reusable setting. Coincidentally, the random oracle paradigm used in their work lends itself nicely to solve the above mentioned issue in the reusable setting. This leads to a natural question of whether this idea alone is sufficient to achieve reusability. Unfortunately, this does not seem to be the case and specifically, the protocol from [IKSS22a] is not reusable.

*Overview of [IKSS22a].* Before we see why the protocol from [IKSS22a] is not reusable, let us first give a high-level overview of this protocol. The protocol is based on the IPS compiler [IPS08] which makes use of three main ingredients. The first, called the *outer protocol*, is a 2-round, 2-client (namely, the receiver and the sender),  $m$ -server MPC protocol for computing the function  $f$ . The outer protocol should be secure against malicious adversaries that corrupt either one of the clients and  $t = \Omega(m)$  servers, and has the following interaction pattern. In the first round, the clients send a message to each one of the servers using their private inputs. The servers perform some local computation on these messages and send the result of this computation to the receiver in the second round. The receiver then decodes these messages to learn the output of  $f$ . The second ingredient, called the *inner protocol*, is a semi-honest secure 2-party protocol for computing the next message function of the servers in the outer protocol. The third ingredient is the *watchlist mechanism* that is implemented using a random oracle. Let now explain how the compiled protocol works.

The sender and the receiver in the compiled protocol generate the first round messages to be sent to each of the servers in the outer protocol. They then start running  $m$  executions of the inner protocol where the  $i$ -th execution is computing the next message function of the  $i$ -th server. The private inputs that the clients use in the  $i$ -th inner protocol execution corresponds to the messages that they send to the  $i$ -th server in the outer protocol. The output of the inner protocol corresponds to the second round messages sent by the servers in the outer protocol and the receiver decodes these messages to learn the output of the functionality. To ensure that a majority of the inner protocol executions are performed correctly, the watchlist mechanism is used. Specifically, the parties after generating their respective messages to each of the  $m$  executions pass these messages to a random oracle that outputs a set  $K$ . The parties send their private inputs and randomness for each inner protocol execution in the set  $K$ . This is verified by the other party. This ensures that a malicious adversary cannot cheat in a large fraction of the inner protocol executions as otherwise the set  $K$  that is output by the random oracle will have a non-empty intersection with the cheating executions. Hence, we can now rely on the security of the outer protocol against a small fraction of the server corruptions to show that the compiled protocol is secure against malicious adversaries.

*Key Challenge.* To make the above construction reusable secure, we need each of the components used in the compiler to be secure in the reusable setting. As discussed earlier, the watchlist mechanism implemented by the random oracle paradigm is serendipitously suitable for the reusable setting. The inner protocol which is only required to be semi-honest secure is also trivially secure in the reusable setting. The key challenge we face is to make the outer protocol secure in the reusable setting.

## 2.1 Constructing a Reusable Outer Protocol

Let us first specify the security properties that a reusable outer protocol needs to satisfy.

*Security Property.* Consider an adversary that corrupts the sender client and a subset of the servers. The honest receiver generates the first round messages to the servers (using its private input) and this message is fixed. The adversary is now allowed to interact with the honest receiver and the servers in many sessions. In each session, the adversary generates a fresh first round sender message to the servers. The honest servers use the fixed receiver’s message and the fresh sender message to compute the second round message in the protocol. The adversary sends an arbitrary second round message from the corrupted servers. It obtains the output computed by the honest receiver in this session and adaptively decides whether to continue with one more session or abort. We require the view of the adversary to be simulatable in the ideal world where the parties have access to the ideal functionality.

*The Case of Constant-Degree Polynomials and Branching Programs.* Before explaining our construction of a reusable outer protocol for computing general circuits, let us first start with a simple case of computing constant degree polynomials. Later, we explain how to extend this construction to securely evaluate branching programs.

Let  $(p_1, \dots, p_\ell)$  be a set of constant-degree polynomials. For the sake of this overview, let us assume that all these polynomials have degree 3. The work of Ishai et al. [IKSS22a] noted that it is not necessary for the outer protocol to satisfy security against stronger malicious adversaries but it is sufficient to start with an outer protocol that is secure against weaker pairwise verifiable adversaries. Pairwise verifiable adversaries are constrained to generate the first round message on behalf of the corrupted clients such that the messages sent to the honest servers pass a pairwise consistency check. Our first observation is that this also extends to the case of reusable security. Specifically, it is sufficient to construct an outer protocol that is reusable secure against pairwise verifiable senders.

Let us first explain the construction of the outer protocol for computing degree-3 polynomials given in [IKSS22a]. In the first round, the clients generate a secret sharing of their private inputs using a 3-multiplicative, pairwise verifiable

secret sharing scheme<sup>7</sup> and send the shares to the servers. The servers then locally compute the degree-3 polynomials on these shares to compute the shares of the outputs. This step relies on the fact that the shares are 3-multiplicative. The servers then send the output shares to the receiver.<sup>8</sup> We note that this protocol is already secure in the reusable setting. This is because the first round message from the receiver to the servers consists of a secret sharing of its private input and this secret sharing can be reused across multiple sessions.

To construct a reusable protocol for securely evaluating branching programs, we make use of randomized encodings [IK00, AIK04]. It is known from these works that branching programs admit a statistically secure degree-3 randomized encoding. Thus, the task of constructing a reusable outer protocol for the case of branching programs reduces to constructing a reusable outer protocol for computing degree-3 polynomials. However, to generate the randomized encoding we need to additionally secret share the randomness used in computing it. A standard way to do this is for the clients to sample randomness  $r_1$  and  $r_2$  respectively and send the shares in the first round. The servers locally compute the shares of  $r_1 + r_2$  and use them to generate the randomized encoding. However, since the first round message from the receiver is fixed once and for all, it means that we need to reuse the receiver’s share of the randomness across multiple sessions. Will this affect security? Fortunately, this does not affect the security as the shares of the output are revealed to the receiver and not to the sender. This means that we can fix  $r_1$  to be the all zeroes string and the sender can be tasked with generating a fresh sharing of the randomness in each session to generate the randomized encoding.

*Extending to Circuits.* All known constructions of randomized encodings for circuits require a PRG [Yao86, IK00, AIK04]. Naïvely incorporating the PRG computation inside the functionality would require non-black-box use of the PRG. Hence, previous NISC protocols for circuits needed to introduce clever mechanisms to ensure that the overall protocol is making black-box use of a PRG. An additional property we need from the outer protocol is that the servers cannot perform any cryptographic operations. This is because the server computations in the IPS compiler are emulated using the inner protocol and if the server computes any cryptographic operations, then functionality that is computed by the inner protocol requires the code of this operation. Therefore, constructions of the outer protocols given in [IPS08, IKSS21, IKSS22a, IKSS22b] required the PRG computations to be done by the clients and the result of these computations to be secret-shared between the servers. Once this is done, the servers can perform a constant degree computation on these shares along with the shares of the input and the randomness to compute a secret sharing of the randomized encoding. Of course, the clients could cheat and send shares of incorrect PRG

---

<sup>7</sup> The standard Shamir secret sharing using bivariate polynomials satisfies this property.

<sup>8</sup> We note that the servers have to additionally re-randomize these shares but we ignore this step to keep the exposition simple.

computations. While there are mechanisms to mitigate this in the single-use setting, unfortunately, this creates serious issues in the reusable setting.

Specifically, consider a malicious adversary that corrupts the sender client and a subset of the servers. The malicious sender client cannot be forced to evaluate the PRGs correctly and hence, could send incorrect sharing of the PRG computations. At a high-level, this means that some entries in the garbled gate table are incorrectly computed. This could force an abort if these particular entries are decrypted in the garbled circuit evaluation. Hence, we need to make sure that the abort event is uncorrelated with the receiver’s input. In the single-use setting this was mitigated using a specific garbled circuit construction due to Beaver et al. [BMR90]. In this construction, the value that is carried by each wire is masked with a random bit and thus, we only decrypt the garbled gate entries corresponding to these masked values. This random masking makes it possible to argue that the abort event is uncorrelated with the receiver’s private input. Unfortunately, in the reusable setting, these masks need to be reused as the receiver’s first round message is fixed across sessions and hence, this offers no security. Thus, we need a brand new approach to prevent such input-dependent aborts in the reusable setting.

*Our Approach: Weakening the Outer Protocol.* This problem seems incredibly hard to solve as there are no black-box mechanisms which can force a malicious client to secret share the correct PRG evaluations. In hindsight, this was also the main reason for why the work of Chase et al. [CDI+19] could not provide a black-box construction for the case of circuits. Instead of dealing with this problem at the outer protocol level, we design new mechanisms to deal with this problem in the protocol compiler. (These mechanisms will only apply to our random oracle based compiler, and do not apply to the “plain” OLE-hybrid model considered in [CDI+19].) Specifically, we consider an outer protocol that is only secure against adversaries that compute the PRGs correctly. We call such adversaries as verifiable adversaries. Next, we give the details about our new protocol compiler that uses this weaker outer protocol to construct a reusable NISC.

## 2.2 A New Protocol Compiler

Our goal is to design a protocol compiler that starts with an outer protocol satisfying reusable security against verifiable adversaries and transforms it into a two-message reusable NISC protocol. In this technical overview, we will only concentrate on proving the reusable security against a malicious sender. Security against malicious receivers follows via standard techniques.

Let us assume that only the sender client needs to compute the PRG evaluations and secret-share them in the outer protocol (in fact, our construction will satisfy this property). Of course, we cannot force the sender client to open all the shares of the PRG computations as this would completely ruin the security of the randomized encoding. Our goal is to design a black-box mechanism that

forces the sender to generate correct sharing of the PRG computations without compromising on the randomized encoding security.

We overcome this by adding one more layer of cut-and-choose. Specifically, instead of emulating one execution of the outer protocol (which consists of  $m$  servers), we emulate  $n$  (for  $n = O(\lambda)$ ) such executions (each containing  $m$  servers). In total, we perform  $n \cdot m$  executions of the inner protocol. Recall that each message sent by the client to a server in the outer protocol consists of two parts: the share of the client's private input and, if the client was the sender, it additionally consists of the share of the PRG evaluation. In each of the  $n$  executions of the outer protocol, we fix the client's shares of the private input to be the same. The sender generates independent PRG evaluations for every execution and generates the shares of these evaluations. If all the emulations are done correctly, then each execution of the outer protocol would be computing a randomized encoding of the function on the same private inputs but using independently chosen random strings. We need to make sure the following two conditions hold: (i) the shares of the private input that the parties use in each execution of the outer protocol are the same, and (ii) the PRG computations and their sharing are performed correctly. Instead of requiring these two conditions to hold exactly, we relax the requirement and ensure that they hold for a large fraction. Specifically, we will make sure that for a large fraction of the servers, the first round messages sent by the malicious sender are the same across all executions and for a large fraction of the executions, the PRG computations and their shares are generated correctly by the sender. We now explain why these two relaxations are sufficient to argue the security of the compiled protocol. The first relaxation does not create any problems we can rely on the security of the outer protocol to additionally corrupt these inconsistent servers (which comprise of a small fraction) and ensure that these inconsistencies do not affect the output obtained by the honest receiver. The second relaxation is a bit more subtle. Note that if the PRG computations are correct, then the receiver's output consists of the evaluation of a properly generated garbled circuit using the same private inputs but using an arbitrary randomness. It follows from the perfect correctness of the garbled circuit evaluation that all these evaluations provide the output of  $f$  applied on the private inputs of the clients. Thus, a majority of these values are going to be the same (corresponding to the correct output) and hence, we can correct the errors caused due to incorrect PRG evaluations by computing the majority function locally on all the  $n$  outputs.

These two relaxations are ensured via two applications of the random oracle based cut-and-choose paradigm. Specifically, we ask the sender to pass its second round message (corresponding to each one of the  $m \cdot n$  executions of the inner protocol) to two random oracles. The first random oracle outputs a subset  $L_1$  of the servers  $[m]$  and the second random oracle outputs a subset  $L_2$  of the executions  $[n]$ . For each server in the set  $L_1$ , the sender client opens up the private input and randomness used in generating the inner protocol messages for this server in each of the  $n$  executions. The honest receiver checks if these messages are correctly generated and if the share of the private input used in each one of the  $n$  executions are identical. For each execution in the set  $L_2$ , the sender

client opens up the shares of the PRG computations sent to all the servers. The receiver checks if the shares correspond to a correct PRG evaluation. The first check ensures that for a majority of the servers, the malicious sender client is using the same share of the private input and these servers are emulated honestly. The second check ensures that except for a small fraction of the executions, the sender client emulates a verifiable adversary and we can rely on the security of the outer protocol against this weaker class to argue the security of the overall protocol. A pictorial representation of the protocol appears in Fig. 2.

*Additional Requirement from the Outer Protocol.* An astute reader who is familiar with the IPS compiler might have noticed the following major challenge in achieving reusable security. An adversarial sender could potentially cheat in a small number of server emulations, such that this number is small enough to escape the watchlist mechanism with non-negligible probability. To be more concrete, assume that the server only cheats in a single execution. Then, the probability that this execution is a part of the watchlist (that is generated using the random oracle) is roughly  $k/m = O(1)$ . Though the number of such cheating sessions are small and are not sufficient to break the privacy of the outer protocol, they could decide if the honest receiver outputs  $\perp$  or obtains the correct output. Hence, in the simulation, it is important to compute the same output that an honest receiver obtains in these cheating server emulations. To achieve this, we corrupt those cheating servers and learn the share that an honest receiver sent to these cheating servers. We then use this share to compute the output that an honest receiver would have obtained by decrypting the cheating sender message. This is possible if the inner protocol satisfied a special property called output equivocation [IKSS22b]. It was recently shown in [IKSS22b] that any NISC protocol with security against malicious senders satisfies output equivocation.

The above simulation technique does not create an issue in the single-use setting. In particular, we can corrupt the servers corresponding to these cheating executions in the outer protocol and obtain the private share sent by the honest receiver and continue with the simulation. However, this causes a serious problem in the reusable setting. Specifically, in each one of the reuse sessions, the adversarial sender client could cheat in a different set of the server executions and cumulatively learn all the private shares of the honest receiver. If this happens, the malicious sender can learn the private input of the receiver in its entirety.

To deal with this issue, we require the outer protocol to satisfy a stronger property called as error correction [IKSS22a]. Informally, this property requires that the output of the receiver’s decoding function depends only on the messages sent from the honest servers and is independent of the messages sent by the corrupt servers. If this property holds, then in each reuse session, we can replace the output of the inner protocol in those cheating executions with a default value and apply the receiver’s decoding function on these outputs. It follows from the error correction property that the output of the honest receiver remains the same after we perform this replacement. This helps in proving that an adversarial sender cannot break receiver privacy by cheating in a different set of executions in each reuse session. We use similar techniques as in [IKSS22a] to add this



extra error correction property. We note that this property was added to the outer protocol in [IKSS22a] to construct a protocol compiler that only makes use of a semi-honest secure inner protocol. However, in our work, we rely on the error correction property to obtain security in the reusable setting.

### 2.3 Extension to the Two-Sided Setting

Let us first state the requirements from a two-sided NISC protocol.

*Two-Sided Reusable NISC.* We say that a NISC protocol is two-sided if the communication channel is bi-directional and the output of  $f$  is delivered to both the parties at the end. In a bit more detail, we model  $f$  as  $(f_0, f_1)$ . For each  $\beta \in \{0, 1\}$ ,  $f_\beta$  takes in offline inputs  $x_0^{\text{off}}$  from  $P_0$ ,  $x_1^{\text{off}}$  from  $P_1$ , a common public online input  $x_{\text{pub}}^{\text{on}}$ , and an online private input  $x_{1-\beta}^{\text{on}}$  from  $P_{1-\beta}$  and delivers  $f_\beta((x_0^{\text{off}}, x_1^{\text{off}}), x_{\text{pub}}^{\text{on}}, x_{1-\beta}^{\text{on}})$  to  $P_\beta$ . The first round message of the protocol depends only on the offline private inputs and the second round message is generated depending on the online inputs. A two-sided NISC protocol is said to be reusable if an adversary corrupts either one of the parties and fixes the first round of interaction once and for all. It then interacts with the other party in multiple sessions. In every session, the honest party generates a second round message using (adaptively chosen) online private inputs and the adversary generates an arbitrary second round message. The adversary learns the output computed by the honest party in this session and adaptively decides whether to continue with one more session or stop. We require the view of the adversary in the real world to be simulatable in an ideal world with access to a trusted functionality that does the following. The parties send their offline inputs to the functionality in the beginning and interact with the functionality in multiple sessions. In every session, the parties send their online inputs to the functionality and it computes the output of  $f$  and delivers the result. We note that this way of modelling the two-sided functionality strictly generalizes the one-sided NISC setting. It also generalizes the prior works on reusable two-round secure computation [BGMM20, BL20, AJJM20, BJKL21, AJJM21, BGSZ22] where the parties commit to their private inputs in the first round and can compute a sequence of functions  $f_i$  on the committed inputs by sending different second round messages. We also note that a stricter model where the functionality takes in private online inputs from both the parties (instead of just one as in our case) is impossible to achieve against rushing adversaries.

*Additional Challenges.* In the two-sided setting, we face some additional challenges. Specifically, we cannot hope to run two instances of the one-sided protocol in the opposite directions to get a two-sided variant. This is because an adversarial client could use two different offline private inputs when acting as the sender and the receiver and learn two different outputs. This will break the security of the two-sided NISC protocol. Therefore, we need an additional mechanism to ensure that the malicious parties are forced to use the same input in those two executions.

*Problem with the Standard Approach.* A standard approach to do this is to give a zero-knowledge proof that the adversary is using the same input in both the sessions, one where it is acting as the sender and the other where it is acting as the receiver. However, as we are interested in giving a black-box construction, we must be careful with the exact zero-knowledge proof that is used.

A black-box way to prove that the adversary used the same offline private input in both the executions is to commit to these two inputs and then prove that the committed values are equal using a black-box commit-and-prove protocol. Such a non-interactive black-box commit-and-prove protocol can be constructed in the random oracle model based on the “MPC-in-the-head” approach of Ishai et al. [IKOS07]. In this approach, the prover generates a secret sharing of the committed values and runs an MPC protocol in its head that reconstructs these two values from the shares and checks equality. It generates the view of each party in the MPC protocol and commits to the view of these virtual parties. The prover then passes these commitments through a random oracle to obtain a set of executions to be opened. The verifier checks if the opened views are consistent and if yes, accepts the proof if the output of the MPC protocol is 1. However, this approach does not directly translate to the reusable setting. This is because the commitments to the offline private input when the honest party acts as the receiver are generated in the first round. In particular, the honest party generates a secret sharing of this private input in the commit-and-prove protocol once and commits to these shares in the first round. For every new second round message in the protocol, we need to generate a fresh secret sharing of the sender offline inputs and prove that these shares correspond to the same value that was used in the receiver side. This means that for such reuse session, we need to generate a fresh proof of consistency and this could imply opening a different subset of the shares of the commitment generated in the first round. After a certain number of reuse sessions, we could open all the shares and this affects the privacy of the honest receiver’s input.

*A Reusable Black-Box Commit-and-Prove.* To deal with this issue, we need a reusable variant of the commit-and-prove protocol. In this variant, the commitments to the secret values are generated once and fixed across multiple sessions. These fixed set of commitments allow a prover to prove in zero-knowledge that these secret values satisfy potentially different predicates in each session. The standard commit-and-prove protocols may not satisfy this reusability property. In this work, we give a construction of a reusable commit-and-prove protocol using additively homomorphic commitments. Specifically, we generate a commitment to the secret values using these homomorphic commitments. For each proof, we use the homomorphism property to generate a fresh secret sharing of the committed values. That is, we generate commitments to randomness and use the additive homomorphism to generate a linear secret sharing of the committed values using the committed randomness. Using these fresh sharings, we can now run an MPC protocol in the head to show that the reconstruction of the newly generated shares satisfy the predicate of interest. Specifically, for each reuse session, we generate a fresh set of secret shares and the problem mentioned above

does not arise. In the full version, we give a construction such a commit-and-prove in the random oracle model (without any additional assumptions) in a weaker setting where the prover and verifier maintain a state that is updated at the end of every proof execution. This construction is based on proactive MPC protocols which allow an adversary to corrupt a different subset of the parties across different time epochs.

*Organization.* In Sect. 3, we give the formal definitions of reusable NISC and reusable two-sided NISC. In Sect. 4, we give the definition of reusable verifiable client-server protocol and we give the construction in full version. In Sect. 5, we give the construction of our black-box reusable NISC protocol. In Sect. 6, we give the definition of a reusable commit-and-prove protocol and give the construction of such a protocol in the full version. In Sect. 7, we state the main theorem regarding construction of our reusable two-sided NISC protocol and give the construction and the proof of security in the full version.

### 3 Definitions

In this section, we give the definition of reusable NISC and its two-sided version.

#### 3.1 Reusable NISC Protocol

Let  $f$  be a two-party functionality between a receiver and a sender. Let  $x$  be the private input of the receiver and  $y$  be the private input of the sender. A NISC protocol<sup>9</sup>  $(\Pi_1, \Pi_2, \text{out}_\Pi)$  between the receiver and the sender is a two-message, malicious-secure protocol that securely computes the ideal functionality  $f$  and delivers the output to the receiver. Specifically, in this protocol,  $\Pi_1$  is run by the receiver using its private input  $x$  to generate the first round message.  $\Pi_2$  is run by the sender on its private input  $y$  and the receiver's first round message to compute the second round message in the protocol.  $\text{out}_\Pi$  is run by the receiver on the sender's message and its private random tape to compute the output of  $f$ . The security is modelled using the standard real-ideal paradigm. For completeness, we provide this definition in the full version.

A reusable NISC protocol is one where the first round message from the receiver is fixed once and for all and the sender can send multiple second round messages (potentially using different inputs). The receiver computes the output of  $f$  on its fixed input and the fresh sender input for each execution. For security, we require this protocol to satisfy standard security against malicious receivers and *reusable* security against malicious senders. In the reusable security game, the adversarial sender is allowed to generate an a priori unbounded polynomial number of second round messages (in an adaptive manner). We now give the formal definition of a reusable NISC protocol.

<sup>9</sup> As our main results are in the random oracle model, we can avoid an explicit setup phase that samples the CRS uniformly and instead use the random oracle's output on some default input as the CRS.

**Definition 1 (Reusable NISC Protocol).** A NISC protocol  $(\Pi_1, \Pi_2, \text{out}_\Pi)$  for computing a two-party function  $f$  is a reusable NISC protocol if it satisfies standard security against malicious receivers and the following reusable sender security. For any PPT adversary  $\mathcal{A}$  that corrupts the sender, there exists a PPT simulator  $\text{Sim}_{\Pi,S}$  such that for all non-uniform PPT (stateful) environments  $\mathcal{Z}$ , the following two distributions are computationally indistinguishable:

- **Real Execution.** The environment  $\mathcal{Z}$  provides the private input  $x$  to the honest receiver and auxiliary input  $z$  to  $\mathcal{A}$ . The honest receiver generates the first round message in the protocol using  $x$  and this message is delivered to  $\mathcal{A}$ . Repeat the following until  $\mathcal{A}$  outputs a special command STOP:
  1.  $\mathcal{Z}$  provides an input  $y$  to the adversary and  $\mathcal{A}$  generates an arbitrary second round message in the protocol.
  2. The honest receiver computes the output of the protocol using  $\text{out}_\Pi$  on the adversarial sender message and its private random tape.
  3. This output is forwarded to  $\mathcal{Z}$  which sends some auxiliary information to  $\mathcal{A}$ .
  4.  $\mathcal{A}$  either outputs STOP or continues to the next iteration.

We call each iteration where the adversary generates a second round message as a session. The output of the real execution corresponds to the output of the honest party in each session and the output of  $\mathcal{A}$  at the end of all sessions.

- **Ideal Execution.** This corresponds to the ideal world interaction where  $\text{Sim}_{\Pi,S}$  and the honest receiver have access a trusted functionality that implements  $f$ . The environment  $\mathcal{Z}$  delivers the private input  $x$  to the honest receiver and auxiliary input  $z$  to  $\text{Sim}_{\Pi,S}$ . The receiver forwards  $x$  to the ideal functionality.  $\text{Sim}_{\Pi,S}$  can interact with the ideal functionality in an a priori unbounded polynomial number of sessions. In each session,
  1.  $\mathcal{Z}$  sends a private input  $y$  to  $\text{Sim}_{\Pi,S}$ .  $\text{Sim}_{\Pi,S}$  sends an arbitrary input to the ideal functionality or a special instruction to the ideal functionality to deliver  $\perp$  to the honest receiver.
  2. The trusted functionality returns the output to the receiver depending on  $\text{Sim}_{\Pi,S}$ 's instruction and this is forwarded to  $\mathcal{Z}$ .
  3.  $\mathcal{Z}$  sends some auxiliary information to  $\text{Sim}_{\Pi,S}$ .
  4.  $\text{Sim}_{\Pi,S}$  decides whether to continue with one more session or stop.
 The output of the ideal execution corresponds to the output of the honest party in each session and the output of  $\text{Sim}_{\Pi,S}$  at the end of all sessions.

### 3.2 Reusable Two-Sided NISC

A two-sided NISC protocol for computing a function  $f = (f_0, f_1)$  is two-round protocol between  $P_0$  and  $P_1$  such that  $P_0$  gets the output of  $f_0$  and  $P_1$  gets the output of  $f_1$ . For each  $\beta \in \{0, 1\}$ ,  $f_\beta$  takes in  $(x_0^{\text{off}}, x_1^{\text{off}})$  which are the offline inputs of the parties, a common public online input  $x_{\text{pub}}^{\text{on}}$ , and a private online input  $x_{1-\beta}^{\text{on}}$  and delivers the output to  $P_\beta$ .

A two-sided NISC protocol is given by a tuple of algorithms  $(\Pi_1, \Pi_2, \text{out}_\Pi)$ .  $\Pi_1$  takes the index  $\beta \in \{0, 1\}$  of the party, its offline private input  $x_\beta^{\text{off}}$  and

produces the first round message sent by  $P_\beta$  which is given by  $\pi_1^{(\beta)}$ .  $\Pi_2$  takes the index  $\beta \in \{0, 1\}$  of the party, the public online input  $x_{\text{pub}}^{\text{on}}$ , the online private input  $x_\beta^{\text{on}}$ , the first round message generated by the other party  $\pi_1^{(1-\beta)}$  and produces the second round message  $\pi_2^{(\beta)}$  of  $P_\beta$ .  $\text{out}_\Pi$  takes in the index  $\beta \in \{0, 1\}$  of the party, its private random tape, and the second round message  $\pi_2^{(1-\beta)}$  generated by  $P_{1-\beta}$  and produces the output of  $f_\beta$  applied on  $((x_0^{\text{off}}, x_1^{\text{off}}), x_{\text{pub}}^{\text{on}}, x_{1-\beta}^{\text{on}})$ . As in the one-sided setting, the security is modelled using the standard real-ideal security paradigm.

We say that a two-sided NISC is reusable if the parties can fix the first round message once and for all and send fresh second round message that depends only on the online private input. The parties use  $\text{out}_\Pi$  to learn the output of the function computed on their fixed offline private inputs and the new online inputs. We require this protocol to satisfy the following security property.

**Definition 2 (Reusable Two-Sided NISC Protocol).** *A two-sided NISC protocol  $(\Pi_1, \Pi_2, \text{out}_\Pi)$  is a reusable NISC protocol for computing  $f = (f_0, f_1)$  if for any PPT adversary  $\mathcal{A}$  that corrupts  $P_{1-\beta}$  for some  $\beta \in \{0, 1\}$ , there exists a PPT simulator  $\text{Sim}_\Pi$  such that for all non-uniform PPT (stateful) environments  $\mathcal{Z}$ , the following two distributions are computationally indistinguishable:*

- **Real Execution.** *For each  $b \in \{0, 1\}$ , the environment delivers the private offline input  $x_b^{\text{off}}$  to  $P_b$  and provides auxiliary input  $z$  to  $\mathcal{A}$ .  $P_\beta$  uses this to generate the first round message in the protocol. The adversary  $\mathcal{A}$  receives this first round message and sends the first round message on behalf of corrupt  $P_{1-\beta}$ . Repeat the following until  $\mathcal{A}$  outputs a special command STOP:*
  1. *The environment  $\mathcal{Z}$  provides an online input  $(x_{\text{pub}}^{\text{on}}, x_b^{\text{on}})$  to  $P_b$  for each  $b \in \{0, 1\}$ .*
  2.  *$P_\beta$  generates the second round message using the online inputs and this is delivered to  $\mathcal{A}$ .  $P_\beta$  then receives the second round message sent by  $\mathcal{A}$ .*
  3. *The honest  $P_\beta$  computes the output of the protocol using  $\text{out}_\Pi$  on the adversarial second round message and its private random tape.*
  4. *The output computed by the receiver is delivered to  $\mathcal{Z}$  who sends some auxiliary information to  $\mathcal{A}$ .*
  5.  *$\mathcal{A}$  either outputs STOP or continues to the next iteration.*

*We call each iteration described above as a session. The output of the real execution corresponds to the output of honest  $P_\beta$  in each session and the output of  $\mathcal{A}$  at the end of all sessions.*

- **Ideal Execution.** *This corresponds to the ideal world interaction where  $\text{Sim}_\Pi$  (corrupting  $P_{1-\beta}$ ) and the honest  $P_\beta$  have access a trusted functionality that implements  $f$ . For each  $b \in \{0, 1\}$ , the environment delivers the private offline input  $x_b^{\text{off}}$  to  $P_b$  and auxiliary input  $z$  to  $\text{Sim}_\Pi$ .  $P_\beta$  sends this to the ideal functionality.  $\text{Sim}_\Pi$  sends an arbitrary offline input on behalf of  $P_{1-\beta}$ .  $\text{Sim}_\Pi$  interacts with the ideal functionality in an a priori unbounded polynomial number of sessions. In each session,*
  1. *The environment delivers an online input  $(x_{\text{pub}}^{\text{on}}, x_b^{\text{on}})$  to  $P_b$  for each  $b \in \{0, 1\}$ .  $P_\beta$  forwards this to the ideal functionality.*

2. The ideal functionality computes  $f_{1-\beta}$  on the fixed offline inputs and the new online input and delivers this output to  $\text{Sim}_\Pi$ .
  3.  $\text{Sim}_\Pi$  can send a special instruction to the ideal functionality to deliver  $\perp$  to the honest receiver or sends an online input  $(\bar{x}_{\text{pub}}^{\text{on}}, x_{1-\beta}^{\text{on}})$ . If  $x_{\text{pub}}^{\text{on}} \neq \bar{x}_{\text{pub}}^{\text{on}}$ , then the trusted functionality delivers  $\perp$  to the receiver. Else, the trusted functionality returns either the output of  $f_\beta$  or  $\perp$  to the honest receiver depending on the instruction from  $\text{Sim}_\Pi$ .
  4. The output delivered to the receiver is forwarded to  $\mathcal{Z}$ .  $\mathcal{Z}$  sends some auxiliary information  $\text{Sim}_\Pi$ .
  5.  $\text{Sim}_\Pi$  then decides to continue with one more execution or stop.
- The output of the ideal execution corresponds to the output of honest  $P_\beta$  in each session and the output of  $\text{Sim}_\Pi$  at the end of all sessions.

## 4 Reusable Verifiable Client-Server Protocol

In this section, we define and construct a reusable verifiable client-server protocol. This protocol will be used as the main building block in the subsequent sections to construct a black-box reusable (two-sided) NISC. We require this protocol to satisfy the following properties.

- **Reusability:** This property requires that the first round message sent by the receiver to be reusable. To be more precise, the receiver sends a single first round message (depending on its private input) to each of the servers and this message is fixed once and for all. The sender can generate multiple (a priori unbounded polynomial number of) first round messages for different choices of its private input. The servers use the fixed first round message from the receiver and the fresh first round message from the sender to compute a second round message in the protocol. The receiver uses this second round message to compute the output of the functionality on its fixed private input and the (fresh) sender input.
- **Error Correction:** Consider an adversary that corrupts the sender and certain number of servers. This property requires that the output of the receiver’s decoding algorithm to remain the same for any choice of second round message sent by the corrupted servers. In other words, the output computed by the receiver is uniquely determined by the messages sent by the honest servers. This property also implies that we can substitute the second round message sent by the adversarial servers with some default values without affecting the receiver’s output.
- **Security against Verifiable Adversaries.** As noted in [IKSS22a], there are barriers in obtaining the error correction property against standard malicious adversaries. Hence, [IKSS22a] defined a weaker class of adversaries called *pairwise verifiable adversaries*. Pairwise verifiable adversaries generate the first round message on behalf of the adversarial client to the honest servers such that it passes some pairwise consistency check. They constructed a protocol that had this error correction property against this weaker class. However,

we are unable to construct a protocol that satisfies both reusability as well as error correction against pairwise verifiable adversaries. Hence, we further weaken the pairwise verifiable adversaries to *verifiable adversaries* which generate the first round message in the protocol in a much more restricted way. Specifically, if the adversary corrupts a sender client then there is a predicate  $P'$  such that the first round messages sent to *all* the honest servers by the adversary satisfy this predicate.<sup>10</sup> In other words, there is some *global predicate*  $P'$  (instead of pairwise local predicate) that the adversarial sender messages must satisfy. On the other hand, if the adversary corrupts the receiver client then the first round messages sent by the receiver should satisfy some pairwise consistency check w.r.t. to a predicate  $P$  (this property is identical to the pairwise verifiable case). It is clear that verifiability restricts the adversarial power even more than pairwise verifiability.

#### 4.1 Definition

We start by describing the syntax of a reusable verifiable client-server protocol.

*Syntax.* A reusable verifiable client-server protocol between two clients, the receiver  $R$  and a sender  $S$  and a set of  $m$  servers is given by a tuple of algorithms  $(\text{Share}_R, \text{ShareInp}_S, \text{ShareRand}_S, \text{Eval}, \text{Dec})$  with the following syntax.<sup>11</sup>

- $\text{Share}_R$  takes the private input  $x$  of the receiver and outputs the first round message  $\{\text{msg}_{R,\text{inp},i}\}_{i \in [m]}$  to be sent to each of the  $m$  servers. Recall that this algorithm is only run once and the messages sent to the servers are reused across different iterations with the sender.
- $\text{ShareInp}_S$  takes the private input  $y$  of the sender and generates  $\{\text{msg}_{S,\text{inp},i}\}_{i \in [m]}$ .  $\text{ShareRand}_S$  takes a uniform random string from the sender and generates  $\{\text{msg}_{S,\text{rand},i}\}_{i \in [m]}$ . The first round message from the sender to the  $i$ -th server consists of  $\{\text{msg}_{S,\text{inp},i}, \text{msg}_{S,\text{rand},i}\}$ . We could have included  $\text{msg}_{S,\text{rand},i}$  as part of  $\text{msg}_{S,\text{inp},i}$  instead of computing it as an output of  $\text{ShareRand}_S$ . However, we choose to split it into two separate algorithms as this presentation is more suitable to be used in our reusable (two-sided) NISC constructions. Looking ahead, we would require the first part of the sender message  $\{\text{msg}_{S,\text{inp},i}\}_{i \in [m]}$  to satisfy local consistency check and the second part  $\{\text{msg}_{S,\text{rand},i}\}_{i \in [m]}$  to satisfy global consistency check (see Footnote 10).
- The  $\text{Eval}$  algorithm takes in the identity  $i$  of the server, the first round messages sent by the clients to this server and outputs the second round message  $\text{msg}_{2,i}$  to be sent to the receiver.
- The  $\text{Dec}$  algorithm takes in  $\{\text{msg}_{2,i}\}_{i \in [m]}$  and computes the output.

<sup>10</sup> We are little imprecise here and this global predicate acts only on a part of the sender's message and not on the whole message. To be more specific, the sender's message consists of two parts. We want the first part to satisfy local consistency and the second part to satisfy global consistency.

<sup>11</sup> We implicitly assume that all the algorithms take in the unary encoding of the security parameter  $1^\lambda$  as part of their inputs.

*Verifiable Adversary.* Before stating the security properties, we start with the definition of a verifiable adversary. A verifiable adversary  $\mathcal{A}$  corrupts either one of the clients and a set  $T$  of the servers. If the adversary corrupts a client  $k \in \{R, S\}$ , then  $\{\text{msg}_{k,\text{inp},i}\}_{i \in [m] \setminus T}$  satisfies a pairwise consistency predicate  $P$ . If  $k = S$ , then we additionally require  $\{\text{msg}_{k,\text{rand},i}\}_{i \in [m] \setminus T}$  to satisfy a global consistency predicate  $P'$ . We note that only the randomness part  $\{\text{msg}_{S,\text{rand},i}\}_{i \in [m] \setminus T}$  is required to satisfy the global consistency predicate and it is sufficient for the input part  $\{\text{msg}_{S,\text{inp},i}\}_{i \in [m] \setminus T}$  to only satisfy a pairwise consistency check. This property will again be crucially used in the construction of a reusable (two-sided) NISC protocol.

**Definition 3 (Pairwise vs Global Predicate).** *Let  $P$  be a pairwise predicate that takes a client index  $k \in \{R, S\}$ , two server indices  $i, j \in [m]$ , the first round message  $(\text{msg}_{k,\text{inp},i}, \text{msg}_{k,\text{inp},j})$  sent by the client  $k$  to the servers  $i$  and  $j$  and outputs  $1/0$ . Let  $P'$  be a global predicate that takes a set  $H \subseteq [m]$ , and the second part of the first round message  $\{\text{msg}_{S,\text{rand},i}\}_{i \in H}$  sent by the sender  $S$  to the servers in  $H$  and outputs  $1/0$ .*

**Definition 4 (Verifiable Adversary).** *An adversary  $\mathcal{A}$  corrupting the client  $k$  and the set  $T$  of the servers is said to be verifiable w.r.t. the pairwise predicate  $P$  and global predicate  $P'$  if it satisfies the following:*

- If  $k \in \{R, S\}$ , then for any two honest servers  $i, j \in [m] \setminus T$ ,  $P(k, i, j, \text{msg}_{k,\text{inp},i}, \text{msg}_{k,\text{inp},j}) = 1$  where  $\text{msg}_{k,\text{inp},i}$  and  $\text{msg}_{k,\text{inp},j}$  are generated by  $\mathcal{A}$  in the protocol execution.
- If  $k = S$ , then the output of the predicate  $P'([m] \setminus T, \{\text{msg}_{S,\text{rand},i}\}_{i \in [m] \setminus T}) = 1$  where  $\{\text{msg}_{S,\text{rand},i}\}_{i \in [m] \setminus T}$  is generated by  $\mathcal{A}$  in the protocol execution.

*Security Definition.* We are now ready to state the security properties that a reusable verifiable client-server protocol needs to satisfy.

**Definition 5 (Reusable Verifiable Client-Server Protocol).** *Let  $f$  be a two-party functionality. A protocol  $\Phi = (\text{Share}_R, \text{ShareInp}_S, \text{ShareRand}_S, \text{Eval}, \text{Dec})$  is a reusable verifiable client-server protocol for computing  $f$  against  $t$  server corruptions if there exists a pairwise predicate  $P$  and a global predicate  $P'$  such that:*

1. **Error Correction:** *Informally, this requires that the output of  $\text{Dec}$  to be uniquely determined by the messages sent by the honest servers. Formally, for any verifiable adversary  $\mathcal{A}$  (see Definition 4) w.r.t.  $P$  and  $P'$  corrupting the sender client  $S$  and a subset  $T$  (where  $|T| \leq t$ ) of the servers and for any two sets of second round messages  $\{\text{msg}_{2,j}\}_{j \in T}$  and  $\{\overline{\text{msg}}_{2,j}\}_{j \in T}$ , we have:*

$$\text{Dec}(\{\text{msg}_{2,j}\}_{j \notin T}, \{\text{msg}_{2,j}\}_{j \in T}) = \text{Dec}(\{\text{msg}_{2,j}\}_{j \notin T}, \{\overline{\text{msg}}_{2,j}\}_{j \in T})$$

where  $\{\text{msg}_{2,j}\}_{j \notin T}$  consists of the second round messages generated by the honest servers (i.e.,  $[m] \setminus T$ ) in the interaction with  $\mathcal{A}$ . In other words, the output of  $\text{Dec}$  remains the same for any choice of second round messages sent by the corrupted servers.



Furthermore, consider a setting where the verifiable adversary  $\mathcal{A}$  generates multiple first round sender messages that all have the same  $\{\text{msg}_{S,\text{inp},j}\}_{j \notin T}$  but potentially different  $\{\text{msg}_{S,\text{rand},j}\}_{j \notin T}$ . Consider the second round messages generated by the servers for each of these sender messages. For each set of these second round server messages (corresponding to each new sender message), we require the output of Dec to be the same. In other words, if a verifiable adversary generates multiple sender messages using the same  $\{\text{msg}_{S,\text{inp},i}\}_{i \notin T}$ , then the output of Dec remains the same.

2. **Security Against Verifiable Receivers:** For any (PPT) verifiable adversary  $\mathcal{A}$  (see Definition 4) w.r.t.  $P$  and  $P'$  corrupting the receiver client and (adaptively) corrupting a set  $T$  of upto  $t$  servers, there exists an (PPT) ideal world simulator  $\text{Sim}_{\Phi,R}$  such that for any choice of private input  $y$  of the honest sender client, the following two distributions are computationally indistinguishable:
  - **Real Execution.** The verifiable adversary  $\mathcal{A}$  interacts with the honest parties (the honest sender client and set of uncorrupted servers) in the protocol. The output of the real execution consists of the output of the verifiable adversary  $\mathcal{A}$ .
  - **Ideal Execution.** This corresponds to the ideal world interaction where  $\text{Sim}_{\Phi,R}$  and the honest sender client have access to the trusted party implementing  $f$ . The honest sender client sends its input  $y$  to  $f$  and  $\text{Sim}_{\Phi,R}$  sends an arbitrary input. The trusted functionality returns the output of  $f$  to  $\text{Sim}_{\Phi,R}$ . The output of the ideal execution corresponds to the output of  $\text{Sim}_{\Phi,R}$ .

3. **Reusable Security against Verifiable Senders:** For any (PPT) verifiable adversary  $\mathcal{A}$  (see Definition 4) w.r.t.  $P$  and  $P'$  corrupting the sender client and a set of servers defined as below, there exists an ideal world (PPT) simulator  $\text{Sim}_{\Phi,S}$  such that for all non-uniform PPT (stateful) environments  $\mathcal{Z}$ , the following two distributions are computationally indistinguishable:
  - **Real Execution.**  $\mathcal{Z}$  delivers the private input  $x$  to the honest receiver and auxiliary input  $z$  to  $\mathcal{A}$ . The receiver uses this private input to generate the first round message in the protocol. The adversary  $\mathcal{A}$  corrupts a set  $T_1$  of the servers and gets the first round messages sent by the honest receiver to  $T_1$ . Repeat the following until adversary  $\mathcal{A}$  outputs a special command STOP:
    - (a)  $\mathcal{Z}$  delivers the private input  $y$  to  $\mathcal{A}$ .  $\mathcal{A}$  adaptively corrupts a set  $T$  of the servers and sends the first round message to the servers  $[m] \setminus (T \cup T_1)$ . Note that adversary does not receive the first round messages sent by the honest receiver to the servers indexed by  $T$ . Further, this set  $T$  could be different across each execution but we require that  $|T \cup T_1| \leq t$ . We additionally require the adversary to be verifiable w.r.t. to the predicates  $P$  and  $P'$  where the set of corrupted servers is given by  $T \cup T_1$ .
    - (b) For each server in  $[m] \setminus (T \cup T_1)$ , we run Eval on the first round message sent by the honest receiver and the first round message sent

- by the adversary in the previous step. The adversary sends arbitrary second round messages from the corrupted servers given by  $T \cup T_1$ .
- (c) We run `Dec` on the second round messages sent by the servers (both honest and the corrupt) and send this output to  $\mathcal{Z}$ .
  - (d)  $\mathcal{Z}$  sends some auxiliary information to  $\mathcal{A}$ .
  - (e)  $\mathcal{A}$  outputs the special symbol `STOP` or decides to continue to the next iteration.

We call each iteration described above as a session. The output of the real execution corresponds to the output of the receiver in each session and the output of  $\mathcal{A}$  at the end of all the executions.

- **Ideal Execution.** This corresponds to the ideal world interaction where  $\text{Sim}_{\Phi,S}$  and the honest receiver client have access to the trusted party that implements  $f$ . The environment delivers an input  $x$  to the receiver and auxiliary input  $z$  to  $\text{Sim}_{\Pi}$ . The receiver sends this to  $f$ .  $\text{Sim}_{\Phi,S}$  interacts with the ideal functionality in an a priori unbounded polynomial number of sessions. In each session,
  - (a)  $\mathcal{Z}$  sends the private input  $y$  to  $\text{Sim}_{\Phi,S}$ .  $\text{Sim}_{\Phi,S}$  sends an arbitrary input to the ideal functionality.
  - (b) The trusted functionality returns the output delivered to the receiver to  $\mathcal{Z}$ .
  - (c)  $\mathcal{Z}$  sends some auxiliary information to  $\text{Sim}_{\Phi,S}$ .
  - (d)  $\text{Sim}_{\Phi,S}$  decides whether to continue with one more execution or stop. The output of the ideal execution corresponds to the output of the receiver in each session and the output of  $\text{Sim}_{\Phi,S}$  at the end of all executions.

We give the construction of reusable outer protocol in the full version.

## 5 Black-Box Reusable NISC

In this section, we give a construction of a black-box reusable NISC protocol. Specifically, we give a black-box transformation from a (non-reusable) NISC protocol to a reusable NISC protocol in the random oracle model. The main theorem we will prove in this section is:

**Theorem 3** *Assume black-box access to a (non-reusable) NISC protocol. Then, there exists a reusable NISC protocol in the random oracle model.*

### 5.1 Construction

We first define a weaker variant of reusable security. In this variant, the reusable security needs to hold only against a weaker class of adversarial senders called as explainable senders [HIK+11]. Intuitively, an explainable sender is required to give an explanation on how it generates the second round message in the protocol. This explanation consists of its private input and the random tape. If this explanation is invalid, we replace the output of the honest receiver with  $\perp$ . We give the formal definition of this variant below.

**Definition 6 (Reusable Security Against Explainable Senders).** *This requirement is the same as the one given in Definition 2 except that in the real execution, the malicious adversary that corrupts the sender has to output an explanation of how it generated the second round message in each iteration. This explanation comprises of its input  $y$  and a random tape  $r$  that it used to generate the second round message. If this explanation is valid, we run the receiver’s output decoding algorithm on the adversarial sender message and provide the output to the adversary. If the explanation is invalid, we replace the output of the receiver in that particular iteration with  $\perp$ .*

We observe that any (non-reusable) NISC protocol satisfies reusable security against explainable senders. This follows directly from the perfect correctness of evaluation algorithm and indistinguishability-based security of the receiver’s message against semi-malicious senders (which is implied by security against malicious senders).

**Proposition 1** *Any NISC protocol satisfies standard security against malicious receivers and reusable security against explainable senders.*

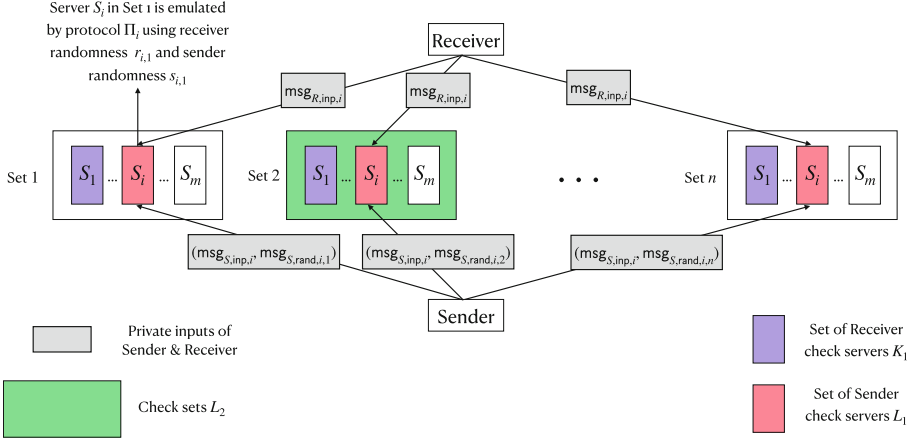
We are now ready to describe our construction.

*Building Blocks.* The construction uses the following building blocks:

1. A reusable verifiable client-server protocol ( $\text{Share}_R, \text{ShareInp}_S, \text{ShareRand}_S, \text{Eval}, \text{Dec}$ ) w.r.t. pairwise predicate  $P$  and global predicate  $P'$  for computing  $f$  against  $t = 4\lambda$  server corruptions (see Definition 5). Let  $m = 20\lambda + 1$  be the number of servers in this protocol (which follows the bounds on the pairwise verifiable 3-multiplicative,  $t$ -error-correctable secret sharing). Our construction given in the full version ensures that  $\text{Eval}$  algorithm does not compute any cryptographic operations.
2. A NISC protocol  $(\Pi_{i,1}, \Pi_{i,2}, \text{out}_{\Pi_i})$  for computing  $\text{Eval}(i, \cdot, \cdot)$  (i.e., the computation done by the  $i$ -th server) for each  $i \in [m]$ . As we are working in the random oracle model, the CRS can be sampled as the output of the random oracle on some default value. From observation 1, we infer that this protocol satisfies reusable security against explainable senders and standard security against malicious receivers.
3. A straight-line extractable non-interactive commitment  $(\text{Com}, \text{Open})$  in the random oracle model (see [Pas03]). We require this commitment to be computationally hiding and statistically binding.
4. Let  $n = 4\lambda$ . Two hash functions  $H_1 : \{0, 1\}^* \rightarrow (\{0, 1\}^{k_m})^m$  and  $H_2 : \{0, 1\}^* \rightarrow (\{0, 1\}^{k_n})^n$  that are modelled as random oracles. Here,  $k_m$  and  $k_n$  are the number of random bits to needed to toss a biased coin that outputs 1 with probability  $p_m = \frac{\lambda}{2m}$  and  $p_n = \frac{\lambda}{2n}$  respectively. We model the output of hash functions  $H_1$  and  $H_2$  as subsets of  $[m]$  and  $[n]$  respectively where each element of the set is included independently with probability  $p_m$  and  $p_n$  respectively.

- **Round-1:** The receiver on private input  $x$  does the following:
  1. It computes  $(\text{msg}_{R,\text{inp},1}, \dots, \text{msg}_{R,\text{inp},m}) \leftarrow \text{Share}_R(x)$ .
  2. For each  $i \in [m]$  and  $j \in [n]$ ,
    - (a) It samples a uniform random tape  $r_{i,j}$  to be used in the protocol  $\Pi_i$ .
    - (b) It computes  $\pi_{i,j,1} := \Pi_{i,1}(\text{msg}_{R,\text{inp},i}; r_{i,j})$ ,  $a_i \leftarrow \text{Com}(\text{msg}_{R,\text{inp},i})$  and  $b_{i,j} \leftarrow \text{Com}(r_{i,j})$ .
  3. It computes  $K_1 = H_1(\text{tag}_R, \{\pi_{i,j,1}, a_i, b_{i,j}\}_{i \in [m], j \in [n]})$  where  $\text{tag}_R \leftarrow \{0, 1\}^\lambda$  and interprets  $K_1$  as a subset of  $[m]$ .
  4. It sends  $(\{\pi_{i,j,1}, a_i, b_{i,j}\}_{i \in [m], j \in [n]}, \text{tag}_R, \{\text{Open}(a_i), \text{Open}(b_{i,j})\}_{i \in K_1, j \in [n]})$  as the first round message.
- **Round-2:** The sender on private input  $y$  does the following:
  1. **Check Phase:**
    - (a) It recomputes  $K_1$  as in step-3 of round-1 and checks if the openings are valid.
    - (b) For each  $i \in K_1$  and for each  $j \in [n]$ , it checks if  $\pi_{i,j,1} = \Pi_{i,1}(\text{msg}_{R,\text{inp},i}; r_{i,j})$ .
    - (c) For each  $i, i' \in K_1$ , it checks if  $\text{msg}_{R,\text{inp},i}$  and  $\text{msg}_{R,\text{inp},i'}$  pass the pairwise consistency check  $P$ .
  2. If any of the above checks fail, it aborts.
  3. Else, it computes  $(\text{msg}_{S,\text{inp},1}, \dots, \text{msg}_{S,\text{inp},m}) \leftarrow \text{ShareInp}_S(x)$ .
  4. For each  $j \in [n]$ , it independently runs  $\text{ShareRand}_S$  to obtain  $(\text{msg}_{S,\text{rand},1,j}, \dots, \text{msg}_{S,\text{rand},m,j})$ .
  5. For each  $i \in [m]$  and  $j \in [n]$ ,
    - (a) It samples a uniform random tape  $s_{i,j}$  to be used in the protocol  $\Pi_i$ .
    - (b) It computes  $\pi_{i,j,2} := \Pi_{i,2}(\pi_{i,j,1}, (\text{msg}_{S,\text{inp},i}, \text{msg}_{S,\text{rand},i,j}); s_{i,j})$  and  $\cdot_{i,j} \leftarrow \text{Com}(\pi_{i,j,2})$ .
    - (c) It computes  $c_i \leftarrow \text{Com}(\text{msg}_{S,\text{inp},i})$ ,  $d_{i,j} \leftarrow \text{Com}(s_{i,j})$ , and  $e_{i,j} \leftarrow \text{Com}(\text{msg}_{S,\text{rand},i,j})$ .
  6. It computes  $L_1 = H_1(\text{tag}_S, \{\cdot_{i,j}, c_i, d_{i,j}, e_{i,j}\}_{i \in [m], j \in [n]})$  and  $L_2 = H_2(\text{tag}_S, \{\cdot_{i,j}, c_i, d_{i,j}, e_{i,j}\}_{i \in [m], j \in [n]})$  where  $\text{tag}_S \leftarrow \{0, 1\}^\lambda$  and interprets  $L_1$  as a subset of  $[m]$  and  $L_2$  as a subset of  $[n]$ .
  7. It sends
    - (a)  $\{\cdot_{i,j}, c_i, d_{i,j}, e_{i,j}\}_{i \in [m], j \in [n]}, \text{tag}_S$ .
    - (b)  $\{\text{Open}(\cdot_{i,j}), \text{Open}(c_i), \text{Open}(d_{i,j}), \text{Open}(e_{i,j})\}_{i \in L_1, j \in [n]}$ .
    - (c)  $\{\text{Open}(e_{i,j})\}_{i \in [m], j \in L_2}$  and  $\{\text{Open}(\cdot_{i,j})\}_{i \in [m], j \notin L_2}$ .
- **Output Computation:** The receiver does the following:
  1. **Check Phase:**
    - (a) It recomputes  $L_1$  and  $L_2$  as in Step-6 of round-2 and checks if all the openings are valid.
    - (b) Using the openings to the commitments  $\cdot_{i,j}$ ,  $c_i$ ,  $d_{i,j}$  and  $e_{i,j}$  given by the sender,
      - i It checks if for each  $i \in L_1$  and  $j \in [n]$  that  $\pi_{i,j,2} = \Pi_{i,2}(\pi_{i,j,1}, (\text{msg}_{S,\text{inp},i}, \text{msg}_{S,\text{rand},i,j}); s_{i,j})$ .
      - ii For each  $i, i' \in L_1$ , it checks if  $\text{msg}_{S,\text{inp},i}$  and  $\text{msg}_{S,\text{inp},i'}$  pass the pairwise consistency check  $P$ .
      - iii For each  $j \in L_2$ , it checks if  $\{\text{msg}_{S,\text{rand},i,j}\}_{i \in [m]}$  pass the global predicate check  $P'$ .
  2. If any of the above checks fail, it aborts.
  3. Else, for each  $j \in [n] \setminus L_2$ ,
    - (a) It computes  $\text{msg}_{2,i,j} \leftarrow \text{out}_{\Pi_i}(\pi_{i,j,2}, r_{i,j})$  for each  $i \in [m]$ .
    - (b) It computes  $\alpha_j = \text{Dec}(\{\text{msg}_{2,i,j}\}_{i \in [m]})$ .
  4. o/p Majority  $(\{\alpha_j\}_{j \in [n] \setminus L_2})$ .

**Fig. 1.** Construction of Reusable Black-Box NISC Protocol



**Fig. 2.** Pictorial Representation of the Protocol. For each  $i \in K_1$ , the receiver opens  $(\text{msg}_{R,\text{inp},i}, \{r_{i,j}\}_{j \in [n]})$ . Similarly, for each  $i \in L_1$ , the sender opens  $(\text{msg}_{S,\text{inp},i}, \{s_{i,j}\}_{j \in [n]})$ . For each  $j \in L_2$ , the sender opens  $(\text{msg}_{S,\text{rand},1,j}, \dots, \text{msg}_{S,\text{rand},m,j})$ .

*Description of Protocol.* The formal description of the protocol is given in Fig. 1. A pictorial representation of our construction is given in Fig. 2.

*Proof of Security.* We defer the proof of security to the full version.

## 6 Non-interactive Reusable Commit-and-Prove

In this section, we define and construct a non-interactive reusable commit-and-prove protocol. This protocol will be used as a key building block in the next section to construct a two-sided reusable NISC protocol.

### 6.1 Definition

*Syntax.* A non-interactive reusable commit-and-prove protocol is given by a tuple of algorithms (Com, Open, Extract, Prove, Verify) with the following syntax.<sup>12</sup>

- **Com** : It takes a message  $x$  as input and outputs a commitment, to this message. We require this commitment to be computationally hiding and statistically binding.
- **Open** : It comprises of the openings to the commitments.
- **Extract** : It takes as input a commitment, and outputs the message inside this commitment.

<sup>12</sup> We implicitly assume that all these algorithms have access to a random oracle and hence, do not include an explicit setup phase. We also assume that all the algorithms take  $1^\lambda$  as an additional input.

- **Prove** : It takes as input a sequence of commitments  $(,_{,1}, \dots, ,_{,n})$ , a function  $f$  and their openings  $(\text{Open}(,_{,1}), \dots, \text{Open}(,_{,n}))$  as input and outputs a proof  $\pi$ .
- **Verify** : It takes a sequence of commitments  $(,_{,1}, \dots, ,_{,n})$ , a function  $f$  and a proof  $\pi$  as input and outputs 1/0 indicating whether the proof is accepting or rejecting.

We now state the properties that such a commit-and-prove protocol must satisfy.

**Definition 7 (Non-Interactive Reusable Commit-and-Prove).** *A tuple of algorithms  $(\text{Com}, \text{Open}, \text{Extract}, \text{Prove}, \text{Verify})$  is said to be a non-interactive reusable commit-and-prove protocol if it satisfies the following properties:*

- $(\text{Com}, \text{Open})$  is a computationally hiding and statistical binding commitment scheme.  $\text{Extract}$  is a straight-line extractor for the commitment scheme.
- **Completeness.** We require that:

$$\Pr[\text{Verify}(X, \text{Prove}(X, \text{Open}(,_{,1}), \dots, \text{Open}(,_{,n}))) = 1] = 1$$

where  $(,_{,1}, \dots, ,_{,n})$  be a sequence of commitments to the messages  $(x_1, \dots, x_n)$ ,  $f$  be a function such that  $f(x_1, \dots, x_n) = 1$  and  $X = (,_{,1}, \dots, ,_{,n}, f)$ .

- **Soundness.** Let  $P^*$  be a non-uniform PPT prover. We require the probability that  $P^*$  wins the following soundness game to be negligible.
  - $(,_{,1}, \dots, ,_{,n}, f, \pi) \leftarrow P^*(1^\lambda)$ .
  - Let  $(x_1, \dots, x_n)$  be the output of  $\text{Extract}$  on inputs  $,_{,1}, \dots, ,_{,n}$  respectively.
  - If  $f(x_1, \dots, x_n) = 0$  and  $\text{Verify}(,_{,1}, \dots, ,_{,n}, f, \pi) = 1$ , then the prover wins this game.
- **Reusable Zero-Knowledge.** There exists a PPT simulator  $\text{Sim}$  such that for every non-uniform PPT verifier  $V^*$ , we have:

$$\text{Real}(V^*) \approx_c \text{Ideal}(\text{Sim}, V^*)$$

where  $\text{Real}$  and  $\text{Ideal}$  experiments are described in Fig. 3.

We defer the construction and proof of security to the full version.

## 7 Black-Box Reusable Two-Sided NISC

In this section, we give a construction of a black-box reusable two-sided NISC. The main theorem we show here is:

**Theorem 4.** *Assume black-box access to:*

1. A (non-reusable) one-sided NISC protocol.
2. A non-interactive reusable commit-and-prove protocol satisfying Definition 7.

Then, there exists a reusable (two-sided) NISC protocol in the random oracle model.

We give the proof of this theorem in the full version.

Real( $V^*$ )	Ideal(Sim, $V^*$ )
1. $(x_1, \dots, x_\ell, n) \leftarrow V^*(1^\lambda)$ .	1. $(x_1, \dots, x_\ell, n) \leftarrow V^*(1^\lambda)$ .
2. $\iota_i \leftarrow \text{Com}(x_i)$ for all $i \in [\ell]$ .	2. $\iota_i \leftarrow \text{Com}(x_i)$ for all $i \in [\ell]$ .
3. Set $\iota_i = \perp$ for all $i \in [\ell + 1, n]$ and $\pi = \perp$ .	3. Set $\iota_i = \perp$ for all $i \in [\ell + 1, n]$ and $\pi = \perp$ .
4. Run until $V^*$ outputs a special symbol STOP :	4. Run until $V^*$ outputs a special symbol STOP :
(a) $(n', x_{\ell+1}, \dots, x_{n'}, f) \leftarrow V^*(\iota_1, \dots, \iota_n, \pi)$ .	(a) $(n', x_{\ell+1}, \dots, x_{n'}, f) \leftarrow V^*(\iota_1, \dots, \iota_n, \pi)$ .
(b) Update the value of $n$ with $n'$ .	(b) Update the value of $n$ with $n'$ .
(c) Compute $\iota_i \leftarrow \text{Com}(x_i)$ for all $i \in [\ell + 1, n]$ .	(c) Compute $\iota_i \leftarrow \text{Com}(x_i)$ for all $i \in [\ell + 1, n]$ .
(d) Set $X = (\iota_1, \dots, \iota_n, f)$ and $w = (\text{Open}(\iota_1), \dots, \text{Open}(\iota_n))$ .	(d) Set $X = (\iota_1, \dots, \iota_n, f)$ and $w = (\text{Open}(\iota_1), \dots, \text{Open}(\iota_n))$ .
(e) If $f(x_1, \dots, x_n) = 1$ , compute $\pi \leftarrow \text{Prove}(X, w)$ .	(e) If $f(x_1, \dots, x_n) = 1$ , compute $\pi \leftarrow \text{Sim}(1^\lambda, X)$ .
5. Output the final view of $V^*$ .	5. Output the final view of $V^*$ .

**Fig. 3.** Descriptions of Real and Ideal experiments.

**Acknowledgments.** Y. Ishai was supported in part by ERC Project NTSC (742754), BSF grant 2018393, ISF grant 2774/20, and a Google Faculty Research Award. D. Khurana was supported in part by DARPA SIEVE award and a gift from Visa Research. A. Sahai was supported in part from a Simons Investigator Award, DARPA SIEVE award, NTT Research, NSF Frontier Award 1413955, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024. A. Srinivasan was supported in part by a SERB startup grant and Google India Research Award.

## References

AIK04. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC<sup>0</sup>. In: 45th FOCS, pp. 166–175. IEEE Computer Society Press, October 2004

AJJM20. Ananth, P., Jain, A., Jin, Z., Malavolta, G.: Multi-key fully-homomorphic encryption in the plain model. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12550, pp. 28–57. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-64375-1\\_2](https://doi.org/10.1007/978-3-030-64375-1_2)

AJJM21. Ananth, P., Jain, A., Jin, Z., Malavolta, G.: Unbounded multi-party computation from learning with errors. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 754–781. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77886-6\\_26](https://doi.org/10.1007/978-3-030-77886-6_26)

- AMPR14. Afshar, A., Mohassel, P., Pinkas, B., Riva, B.: Non-interactive secure computation based on cut-and-choose. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 387–404. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-642-55220-5\\_22](https://doi.org/10.1007/978-3-642-55220-5_22)
- ASH+20. Abascal, J., Sereshgi, M.H.F., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Is the classical GMW paradigm practical? The case of non-interactive actively secure 2PC. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1591–1605. ACM (2020)
- BGMM20. Bartusek, J., Garg, S., Masny, D., Mukherjee, P.: Reusable two-round MPC from DDH. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12551, pp. 320–348. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-64378-2\\_12](https://doi.org/10.1007/978-3-030-64378-2_12)
- BGSZ22. Bartusek, J., Garg, S., Srinivasan, A., Zhang, Y.: Reusable two-round MPC from LPN. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) KC, vol. 13177, LNCS, pp. 165–193. Springer, Cham (2022). [https://doi.org/10.1007/978-3-030-97121-2\\_72022](https://doi.org/10.1007/978-3-030-97121-2_72022)
- BJKL21. Benhamouda, F., Jain, A., Komargodski, I., Lin, H.: Multiparty reusable non-interactive secure computation from LWE. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 724–753. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77886-6\\_25](https://doi.org/10.1007/978-3-030-77886-6_25)
- BL20. Benhamouda, F., Jain, A., Komargodski, I., Lin, H.: Multiparty reusable non-interactive secure computation from LWE. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 724–753. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77886-6\\_25](https://doi.org/10.1007/978-3-030-77886-6_25)
- BMR90. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC, pp. 503–513. ACM Press, May 1990
- CCKM00. Cachin, C., Camenisch, J., Kilian, J., Müller, J.: One-round secure computation and secure autonomous mobile agents. In: Montanari, U., Rolim, J.D.P., Welzl, E. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 512–523. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-45022-X\\_43](https://doi.org/10.1007/3-540-45022-X_43)
- CDI+19. Chase, M., Dodis, Y., Ishai, Y., Kraschewski, D., Liu, T., Ostrovsky, R., Vaikuntanathan, V.: Reusable non-interactive secure computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 462–488. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-26954-8\\_15](https://doi.org/10.1007/978-3-030-26954-8_15)
- DILO22. Dittmer, Y.I., Lu, S., Ostrovsky, R.: Authenticated garbling from simple correlations. IACR Cryptol. ePrint Arch., page 836 (2022)
- DIO21. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-point zero knowledge and its applications. In: Tessaro, S. (ed.) ITC 2021, vol.199 of LIPIcs, pp. 5:1–5:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
- HIK+11. Haitner, I., Ishai, Y., Kushilevitz, E., Lindell, Y., Petrank, E.: Black-box constructions of protocols for secure computation. SIAM J. Comput. **40**(2), 225–266 (2011)
- HK07. Horvitz, O., Katz, J.: Universally-composable two-party computation in two rounds. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 111–129. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-74143-5\\_7](https://doi.org/10.1007/978-3-540-74143-5_7)



- IK00. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: 41st FOCS, pp. 294–304. IEEE Computer Society Press, November 2000
- IKO+11. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 406–425. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-20465-4\\_23](https://doi.org/10.1007/978-3-642-20465-4_23)
- IKOS07. 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
- IKSS21. Ishai, Y., Khurana, D., Sahai, A., Srinivasan, A.: On the round complexity of black-box secure MPC. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 214–243. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84245-1\\_8](https://doi.org/10.1007/978-3-030-84245-1_8)
- IKSS22a. Ishai, Y., Khurana, D., Sahai, A., Srinivasan, A.: Round-optimal black-box protocol compilers. In: Dunkelman, O., Dziembowski, S. (eds) EUROCRYPT 2022, Part I, vol. 13275 of LNCS, pp. 210–240. Springer, Heidelberg, May/June 2022. [https://doi.org/10.1007/978-3-031-06944-4\\_8](https://doi.org/10.1007/978-3-031-06944-4_8)
- IKSS22b. Ishai, Y., Khurana, D., Sahai, A., Srinivasan, A.: Round-optimal black-box secure computation from two-round malicious OT. In: TCC (2022)
- IPS08. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer – efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-85174-5\\_32](https://doi.org/10.1007/978-3-540-85174-5_32)
- MR17. Mohassel, P., Rosulek, M.: Non-interactive secure 2PC in the offline/online and batch settings. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10212, pp. 425–455. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-56617-7\\_15](https://doi.org/10.1007/978-3-319-56617-7_15)
- Pai99. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT’99. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
- Pas03. Pass, R.: On deniability in the common reference string and random oracle model. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 316–337. Springer, Heidelberg (2003). [https://doi.org/10.1007/978-3-540-45146-4\\_19](https://doi.org/10.1007/978-3-540-45146-4_19)
- Yao86. Chi-Chih Yao. A.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press, October 1986



# Maliciously-Secure MrNISC in the Plain Model

Rex Fernando<sup>1</sup>, Aayush Jain<sup>1</sup>, and Ilan Komargodski<sup>2</sup>

<sup>1</sup> Carnegie Mellon University, Pittsburgh, PA, USA  
rex1fernando@gmail.com, aayushja@andrew.cmu.edu

<sup>2</sup> School of Computer Science and Engineering, Hebrew University of Jerusalem  
and NTT Research, Jerusalem 91904, Israel  
ilank@cs.huji.ac.il

**Abstract.** We study strong versions of round-optimal MPC. A recent work of Benhamouda and Lin (TCC '20) identified a version of secure multiparty computation (MPC), termed *Multiparty reusable Non-Interactive Secure Computation* (MrNISC), that combines at the same time several fundamental aspects of secure computation with standard simulation security into one primitive: round-optimality, succinctness, concurrency, and adaptivity. In more detail, MrNISC is essentially a two-round MPC protocol where the first round of messages serves as a reusable commitment to the private inputs of participating parties. Using these commitments, any subset of parties can later compute any function of their choice on their respective inputs by broadcasting one message each. Anyone who sees these parties' commitments and evaluation messages (even an outside observer) can learn the function output and nothing else. Importantly, the input commitments can be computed without knowing anything about other participating parties (neither their identities nor their number) and they are reusable across any number of computations.

By now, there are several known MrNISC protocols from either (bilinear) group-based assumptions or from LWE. They all satisfy semi-malicious security (in the plain model) and require trusted setup assumptions in order to get malicious security. We are interested in *maliciously secure MrNISC protocols in the plain model, without trusted setup*. Since the standard notion of polynomial simulation is un-achievable in less than four rounds, we focus on security with *super-polynomial*-time simulation (SPS).

Our main result is the first maliciously secure SPS MrNISC in the plain model. The result is obtained by generically compiling any semi-malicious MrNISC and the security of our compiler relies on several well-studied assumptions of an indistinguishability obfuscator, DDH over  $\mathbb{Z}_p^*$  and asymmetric pairing groups, and a time-lock puzzle (all of which need to be sub-exponentially hard). As a special case, we obtain the first 2-round maliciously secure SPS MPC based on well-founded assumptions. This MPC is also concurrently self-composable and its first message is short (i.e., its size is independent of the number of the participating parties) and reusable throughout any number of computations. Prior to our work, for two round maliciously secure MPC, neither concurrent MPC

nor reusable MPC nor MPC with first message independent in the number of parties was known from any set of assumptions. Of independent interest is one of our building blocks: the first construction of a one-round non-malleable commitment scheme from well-studied assumptions, avoiding keyless hash functions and non-standard hardness amplification assumptions. The full version of this paper can be found at [26].

## 1 Introduction

In this work, we study the round complexity of cryptographic protocols, giving special attention to secure multi-party computation (MPC). MPC allows a group of mutually distrusting parties  $P_1, \dots, P_n$ , each with private input  $x_i$ , to compute the evaluation of some function  $f(x_1, \dots, x_n)$  without revealing their inputs to each other [12, 21, 30].

Round complexity is a fundamental measure of both the efficiency and power of cryptographic protocols. The importance of this measure is strongly grounded in practice: while the bandwidth of modern networks has constantly been increasing, there is a physical lower bound on their latency, imposed by distance and the speed of light. The round complexity of a protocol can also affect its security properties. One very useful property of fully non-interactive and quasi-non-interactive<sup>1</sup> arguments is that proofs can be posted to some public bulletin board, like a blockchain, and then any party can later independently verify its validity, even if the original prover is offline. This enables arguments to be recursively composed, which has been used to achieve fundamental new results in the areas of succinct arguments [15], and also to achieve new space and communication efficient secure multi-party computation protocols [25].

The round complexity of MPC protocols in particular has been well-studied over the last few decades. The original MPC construction of [30] was highly round-inefficient, taking a number of rounds proportional to the depth of the circuit for the functionality being computed. Since then, a long line of work [2, 11, 19, 22, 23, 29, 34, 35, 42] has made dramatic improvements, with recent works finally achieving four rounds [2, 19, 22, 23]. This was shown to be optimal by the works of [29, 34], which showed that achieving secure computation in three rounds within the standard regime of black-box polynomial-time simulation is impossible.

In the classical definition of simulation security for MPC protocols, the parties are assumed to run the protocol in an isolated environment, separate from other parties and other executions of protocols. While this definition is simple and elegant, the ubiquity of the internet means that this assumption is not very realistic. The notion of *concurrent security* fixes this by allowing an adversary to spawn an arbitrary number of parties and executions of a protocol. Unfortunately, the work of [8] showed that concurrent security is impossible *in any number of rounds* within the standard regime of black-box polynomial-time simulation.

---

<sup>1</sup> By *quasi-non-interactive* we refer to “non-interactive” protocols that require a trusted setup such as a common reference string.

The exciting work of [40] introduced a very useful relaxation of standard polynomial-time simulation, called *super-polynomial-time simulation* (SPS). In this new definition, the simulator is allowed to run for slightly longer than polynomial-time. This has been used, among other things, to achieve concurrent security for MPC protocols by the works of [20, 28, 37], sidestepping the impossibility result of [8]. In 2017, the work of [7] constructed a concurrent MPC protocol in three rounds, thus bypassing both the lower bounds of [29, 34] and [8] at once. For several years, this has been the state of the art in terms of the round complexity of both MPC and concurrent-secure MPC in the plain model. A very recent work [1] partially advanced the state of the art in terms of round complexity, giving a two-round standalone-secure MPC protocol in the plain model. However, their security proof relies on ad-hoc (exponentially strong) assumptions that are novel to their work, and they do not achieve concurrent security.<sup>2</sup>

An important question, then, is whether concurrent-secure MPC, or even standalone MPC, can be achieved in two rounds in the plain model, without setup, relying on well-studied assumptions. In this work, we study this question.

**MrNISC.** Going one step further, it is natural to ask whether MPC can be done in one round, with each party sending a single simultaneous message. However, one can very easily show that this is impossible, via the following argument, commonly referred to as the *residual function attack*. Consider the case of two parties  $P_1$  and  $P_2$ , and say that  $P_1$  sends its message  $m_1$ . Then  $P_2$  should be able to compute and send her message  $m_2$ , so that both parties learn  $f(x_1, x_2)$ . However, this means that  $P_2$  can compute  $m'_2$  for any other  $x'_2$  in her head, and learn  $f(x_1, x_2)$  as well. She can do this for arbitrarily many  $x'_2$ . This means that parties are able to learn much more than is allowed by a secure MPC protocol. This simple argument also extends to the case of protocols with trusted setup, showing that one-round protocols are also impossible in this case.

This raises the question, how close can we get to a non-interactive protocol without running into this impossibility? We study this question via a recent new strong version of MPC, identified by a recent work by Benhamouda and Lin [14] and termed *Multiparty reusable Non-Interactive Secure Computation* (MrNISC). MrNISC requires the following general structure:

1. *Input encoding*: at any time, a party can publish an encoding of its input noninteractively, independent of the number of parties.
2. *Computation encoding*: At any time, any subset  $I$  of parties can jointly compute a function  $f$  on their inputs  $x_I = \{x_i\}_{i \in I}$  by broadcasting a single public message. Each party's message is only dependent on the input encodings of the parties in  $I$ .

Parties are allowed to join the system at any time by publishing their input encoding, even after an arbitrary number of computation sessions have occurred.

---

<sup>2</sup> We discuss this work further in Sect. 1.3.

In this way, MrNISC achieves essentially the best-possible form of non-interactivity for MPC protocols without running into the aforementioned impossibility: once parties have committed to their input, any subset of parties can compute an arbitrary function on their committed inputs via a single round. Note that MrNISC is a strict generalization of two-round concurrent-secure MPC.

Several MrNISC protocols have been constructed in the *semi-malicious* regime, where security only holds for adversaries who follow the protocol specification.<sup>3</sup> Benhamouda and Lin [14] constructed such a protocol for all efficiently computable functionalities relying on the DDH assumption in asymmetric bilinear groups. In two concurrent follow-up works, Ananth et al. [3] and Benhamouda et al. [13] obtained MrNISC protocols relying on Learning With Errors (LWE). However, it was unknown whether it is possible to construct MrNISC in the plain model which satisfies the full malicious version of security, where adversaries can deviate arbitrarily from the protocol specification.

## 1.1 Our Results

In this paper, we give the first affirmative answer to the above question. Specifically, relying on commonly-used, well-established assumptions, we obtain a maliciously secure SPS MrNISC in the plain model, without any trusted setup. In particular, this implies a concurrently secure SPS MPC in two rounds from the same assumptions. We state our (informal) theorem below.

**Theorem 1.1 (Main Result, informal).** *Assume the existence of an indistinguishability obfuscation (iO) scheme which is subexponentially-secure, subexponential DDH (over both asymmetric pairing groups<sup>4</sup> and  $\mathbb{Z}_p^*$ ), and subexponential time-lock puzzles. Then there exists a malicious-secure MrNISC in the plain model, with a super-polynomial simulator.*

**Key ideas.** Our result is obtained via a generic compiler which takes any subexponentially-secure semi-malicious secure MrNISC and upgrades it to malicious security. As mentioned above, the work of [14] showed that such a semi-malicious-secure MrNISC exists assuming subexponential DDH over asymmetric pairing groups. Our transformation relies heavily on the idea of multiple *axes of hardness* [38], where there are multiple ways to measure the hardness of a problem, such as circuit size and circuit depth. This allows one to define pairs of problems  $(A, B)$  where  $A$  is simultaneously harder than  $B$  (with respect to one axis) and easier than  $B$  (with respect to the other). Time-lock puzzles are a well-known way to achieve such scenarios based on circuit size and depth.

**Implications for (Classical) MPC.** As mentioned, it is possible to view an MrNISC as a standard MPC. Specifically, we get the following:

<sup>3</sup> Semi-malicious security allows the adversary to choose arbitrary randomness for the parties, but otherwise requires honest behavior.

<sup>4</sup> DDH assumption over asymmetric pairing groups is also referred to as the SXDH assumption. We will interchangeably use SXDH wherever we specifically require DDH over asymmetric pairing groups.

- Our MrNISC implies the first **concurrent** two-round maliciously secure SPS MPC. Indeed, at any point in time, parties can join the protocol by publishing their input encodings and even start evaluation phases. This could happen even after some of the other parties published their input encodings and participated in several evaluation phases. The only previously known *malicious* (SPS) concurrent MPC required three rounds [7].
- Our MrNISC implies the first 2-round maliciously secure SPS MPC with a **short and reusable first message**, based on any assumption. Namely, the first round message is not only independent of the function to be computed (which is necessary for reusability), but it is actually generated independently of the number of participating parties. All prior MPC protocols with this property only satisfy semi-malicious security in the plain model [3,9,10,13,14].
- Our MrNISC implies the first 2-round maliciously secure SPS MPC based on **well-studied, falsifiable assumptions**.

### Notable Building Blocks

In the course of obtaining our main result, we achieve two intermediate results, in the areas of zero-knowledge and non-malleable commitments.

First, we give a new definition of two-round zero knowledge, called *reusable statistical zero-knowledge with sometimes-statistical soundness*. This new type of argument that satisfies both statistical zero knowledge and a weakened form of statistical soundness. (Note that it is well-known that achieving both statistical zero knowledge and full statistical soundness is impossible for all statements in NP unless the polynomial-time hierarchy collapses [41].) We also require a strong form of reusability. We show the following:

**Theorem 1.2 (Informal).** *Assume the existence of a subexponentially-secure indistinguishability obfuscation (iO) scheme, subexponential DDH (over both  $\mathbb{Z}_p^*$  and asymmetric pairing groups), and subexponential time-lock puzzles. Then there exists a reusable statistical ZK argument with sometimes-statistical soundness as defined in Definition 5.4.*

Second, we give a new one-round non-malleable commitment in the simultaneous-message model under better assumptions than were previously known. This commitment satisfies a strong definition of security called CCA-non-malleability. We prove the following theorem:

**Theorem 1.3 (Informal).** *Assume the existence of a subexponentially-secure indistinguishability obfuscation (iO) scheme, subexponential SXDH, and subexponential time-lock puzzles. Then, there exists a subexponentially-secure one-round CCA commitment scheme supporting a super-polynomial number of tags.*

Non-interactive non-malleable commitments were first constructed by the work of [39], using very strong and non-standard assumptions. In particular, their assumption incorporates a strong form of non-malleability into it. The

works of [18, 27] were able to obtain constructions based on different assumptions, including (among other things) a rather new assumption called *keyless multi-collision-resistant hash functions* [16]. This assumption, which is described in more detail below, is still somewhat strong as we do not have any instantiation of it besides using cryptographic hash functions. In contrast, our commitment scheme relies solely on well-established assumptions which have a long history of study.

Our construction is based heavily on and improves upon the work of [36], which achieves a weakened version of one-round non-malleable commitments. In order to achieve our main result, we need full CCA-non-malleable commitments which work in one round, so the construction of [36] will not suffice as-is. We elaborate on this in Sect. 2.

## Putting Things Together

Our compiler makes use of these two new tools in order to upgrade security of a semi-malicious MrNISC scheme. Informally, we achieve the following:

### Theorem 1.4 (The Compiler, Informal).

*Assume the existence of subexponential variants of the following:*

- *a reusable two-round statistical zero knowledge argument with sometimes-statistical soundness,*
- *a one-round non-malleable commitment,*
- *a non-interactive perfectly-binding commitment,*
- *a pseudorandom function,*
- *a witness encryption scheme for NP,*
- *and finally, a semi-malicious MrNISC scheme.*

*Then, there exists a malicious-secure MrNISC scheme in the plain model, with super-polynomial simulation.*

## 1.2 On the Necessity of iO

We make use of an obfuscation scheme when constructing both our zero knowledge scheme as well as our non-malleable commitment scheme. Also, it is directly used to get the witness encryption scheme. We do not know if iO can be avoided in constructing MrNISC in the plain model.

As mentioned above, constructions of one-round non-malleable commitments exist from other assumptions than iO [18, 39], however these constructions rely on assumptions that are problematic for various reasons. The only known route to avoid these assumptions is via iO [36] but even then previous work failed to achieve one-round protocols.

We now discuss the need in a witness encryption scheme. Intuitively, it seems that some sort of witness encryption for a specific language is required when upgrading security for a semi-malicious MrNISC scheme in the plain model, for the following reason. Since one-round zero knowledge is impossible without

setup [31], honest parties are forced to send their second-round semi-malicious MrNISC messages without knowing whether the first round is honest. Sending these messages in the clear would violate security, so the parties must somehow send a “locked” version of their second-round such that they are only revealed conditioned on the first round being honest. Since these messages must be publicly unlockable, this means that the second round is some form of witness encryption. We explain this in more detail in Sect. 2. It is an interesting open question whether it is possible to build a witness encryption scheme for this specific type of statement without relying on iO.

### 1.3 Related Work

A recent work of Agarwal, Bartusek, Goyal, Khurana, and Malavolta [1] gave the first two-round standalone maliciously secure MPC in the plain model. Although an exciting first step, the result is nonstandard in several ways. First, they require the existence of several primitives (including semi-malicious MPC) which are *exponentially secure* in the number of parties. Their construction also requires a special type of non-interactive non-malleable commitment. Notably, neither the non-interactive commitments of [18, 32] nor the weakly non-interactive commitments of [36], nor our new one-round non-malleable commitment scheme can be used to instantiate this (because they strongly rely on *exponential* full security and non-interactivity). The authors of [1] propose two instantiations which work for their construction. One instantiation relies on factoring-based *adaptive* one-way functions [39],<sup>5</sup> a strong assumption that incorporates a strong non-malleability flavor. Another instantiation relies on an exponential variant of the “hardness amplifiability” assumption of [18], along with keyless multi-collision resistant hash functions [17]. Both of these assumptions are still highly non-standard:

1. A keyless multi-collision resistant hash function is a single publicly known function for which (roughly) collisions are “incompressible”, namely, it is impossible to encode significantly more than  $k$  collisions using only  $k$  bits of information. While keyless hash functions are formally a plain-model assumption, there is no known plain-model instantiation based on standard assumptions. The only known instantiation is either in the random oracle model, or by heuristically assuming that some cryptographic hash function, like SHA-256, is such.
2. Hardness amplification assumptions postulate (roughly) that the XOR of independently committed random bits cannot be predicted with sufficiently large advantage. There are concrete (contrived) counter examples for this type of assumptions showing that they are generically false [24], although they certainly might hold for specific constructions.

---

<sup>5</sup> An adaptive one-way function is a non-falsifiable hardness assumption postulating the existence of a one-way function  $f$  that is hard to invert on a random point  $y = f(x)$  even if you get access to an inversion oracle that inverts it on every other point  $y' \neq y$ .



The specific variant used by Agarwal et al. is novel to their work. It assumes *exponential* hardness amplification against PPT adversaries, i.e., that there exists a constant  $\delta > 0$  such that for large enough  $\ell$ , the XOR of  $\ell$  independently committed random bits cannot be predicted by a PPT adversary with advantage better than  $2^{-\ell\delta}$ . This assumption (similarly to [39]’s adaptive one-way functions) also incorporates a non-malleability flavor.

Because of this, there is no way to instantiate the protocol of [1] relying on any well-studied assumptions, or even on assumptions not specifically formulated in order to achieve non-malleable commitments. These drawbacks unfortunately seem inherent in the techniques used by [1]. Our work uses a completely different approach from their work, and is thus able to achieve a strictly stronger result, without using ad-hoc assumptions.

## 2 Technical Overview

In this section, we give an overview of our constructions and the main ideas needed to prove their security. We start by reviewing the syntax of MrNISC, as defined by Benhamouda and Lin [14].

**Model and syntax.** A MrNISC consists of an input encoding phase done without coordination with other parties in the system (i.e., without even knowing they exist), and an evaluation phase in which only relevant parties participate by publishing exactly one message each. In other words, MrNISC is a strict generalization of 2-round MPC with the following properties:

- there is no bound on the number of parties;
- multiple evaluation phases can take place with the same input encodings;
- parties can join at any point in time and publish their input encoding, even after multiple evaluation phases occurred.

We assume all parties have access to a broadcast channel that parties use to transmit messages to all other parties. The formal syntax of an MrNISC consists of three polynomial-time algorithms (**Encode**, **Eval**, **Output**), where **Encode** and **Eval** are probabilistic, and **Output** is deterministic. The allowed operations for a party  $P_i$  are:

- **Input Encoding phase:** each party  $P_i$  computes  $m_{i,1}, \sigma_{i,1} \leftarrow \text{Encode}(1^\lambda, x_i)$ , where  $x_i$  is  $P_i$ ’s private input,  $m_{i,1}$  is  $P_i$ ’s round 1 message, and  $\sigma_{i,1}$  is  $P_i$ ’s round 1 private state. It broadcasts  $m_{i,1}$  to all other parties.
- **Function Evaluation phase:** any set of parties  $I$  can compute an arity- $|I|$  function  $f$  on their respective inputs as follows. Each party  $P_i$  for  $i \in I$  computes  $m_{i,2} \leftarrow \text{Eval}(f, \sigma_{i,1}, I, \{m_{j,1}\}_{j \in I})$ , where  $f$  is the function to compute,  $x_i$  is  $P_i$ ’s private input,  $\sigma_{i,1}$  is the private state of  $P_i$ ’s input encoding,  $\{m_{j,1}\}_{j \in I}$  are the input encodings of all parties in  $I$ , and the output  $m_{i,2}$  is  $P_i$ ’s round 2 message. It broadcasts  $m_{i,2}$  to all parties in  $I$ .

- **Output phase:** upon completion of the evaluation phase by each of the participating parties, anyone can compute  $y \leftarrow \text{Output}(\{m_{i,1}, m_{i,2}\}_{i \in I})$  which should be equal to  $f(\{x_j\}_{j \in I})$ .

**Security.** For security, we require that an attacker does not learn any information beyond what is absolutely necessary, which is the outputs of the computations. Formally, for every “real-world” adversary that corrupts the evaluator and a subset of parties, we design an “ideal world” adversary (called a simulator) that can simulate the view of the real-world adversary using just the outputs of the computations. As in all previous works on MrNISC (including [3, 13, 14]), we assume static corruptions, namely that the adversary commits on the corrupted set of parties at the very beginning of the game. However, all previous works only achieved semi-malicious security (unless trusted setup assumptions are introduced). This notion of security, introduced by Asharov et al. [4], only considers corrupted parties that follow the protocol specification, except letting them choose their inputs and randomness arbitrarily. In contrast, we consider the much stronger and more standard notion of *malicious* security, which allows the attacker to deviate from the specification of the protocol arbitrarily.

More precisely, in malicious security, the adversary can behave arbitrarily in the name of the corrupted parties. Specifically, after the adversary commits on the corrupted set of parties, it can send an arbitrary round 1 message for a corrupted party, ask for a round 1 message of any honest party (with associated private input), ask an honest party to send the round 2 message corresponding to an evaluation of an arbitrary function on the round 1 message of an arbitrary set of parties, and send an arbitrary round 2 message of a malicious party corresponding to an evaluation of an arbitrary function on the round 1 message of an arbitrary set of parties. The simulator needs to simulate the adversary’s view with the assistance of an ideal functionality that can provide only the outputs of the computations that are being performed throughout the adversary’s interaction.

Typically, protocols are called *maliciously secure* if for every polynomial-time adversary, there is a polynomial-time simulator for which the real-world experiment and the ideal-world experiment from above are indistinguishable. However, as mentioned, it is impossible to achieve such a notion of malicious security for MPC (let alone MrNISC) in merely two rounds unless trusted setup assumptions are introduced. Therefore, we settle for super-polynomial time simulation (SPS), which means that the simulator can run in super-polynomial time. In contrast, the adversary is still assumed to run in polynomial time.

We refer to Sect. 4 for the precise definition.

**Terminology.** For the sake of brevity, we will sometimes refer to the *input encoding phase* as *round 1*, and the *function evaluation phase* as *round 2*.

## 2.1 The MrNISC Protocol

To obtain our main result, we will start with a semi-malicious-secure MrNISC protocol [13, 14] and introduce modifications to achieve malicious security. Recall that semi-malicious security only guarantees security when the adversary follows the honest protocol specification exactly, except that it can arbitrarily choose corrupted parties’ randomness. We would like to use the following high-level approach used by many classical MPC protocols. During the input encoding phase, we require each party to commit to its input and randomness in addition to publishing a semi-malicious input encoding, and then to prove using zero-knowledge that all of its semi-malicious MrNISC messages were generated by following the prescribed protocol using that committed input and randomness. However, a problem arises when using this strategy with 2-round protocols. (Note that MrNISC requires that evaluation can be carried out in two rounds; in this way, it is a strict generalization of 2-round MPC.) This problem comes from the fact that zero-knowledge in the plain model requires at least two rounds. Assuming we use such a 2-round ZK scheme, honest parties would need to send their second-round MrNISC messages before finding out whether the first-round MrNISC messages were honest. This completely breaks security—if any party publishes semi-malicious messages based on a non-honest transcript, the semi-malicious protocol can make no security guarantees about these messages.

We need some way of overcoming this problem. That is, we need a way to publish second-round messages so that they are only revealed if the first round is honest. To this end, we are going to use *witness encryption* as a locking mechanism: we “lock” the round 2 message of the underlying (semi-malicious) MrNISC and make sure that it can be unlocked only if all involved parties’ proofs verify. More precisely, party  $i$  does:

1. *Round 1 message*: Commit to its input and randomness and publish a round 1 message using the underlying MrNISC with the committed input/randomness pair. At the same time, generate a verifier’s first-round ZK message for the other parties.
2. *Round 2 message*: Compute a round 2 message using the underlying MrNISC with randomness derived from the secret state. Generate a zero-knowledge proof that this was done correctly. Publish a witness encryption hiding the aforementioned round 2 message that could be recovered by supplying valid proofs that all other parties’ first-round messages were created correctly.

With this template in mind, even before starting to think about what a security proof will look, it is already evident that there are significant challenges in realizing the building blocks. Here are the three main challenges.

**Challenge 1: The ZK argument system.** The first challenge arises from trying to use ZK arguments as witnesses for the witness encryption scheme. Recall that witness encryption allows an encryptor to encrypt a message with respect to some statement  $\Phi$ , and only if  $\Phi$  is false, then the message is hidden. Witness encryption (WE) crucially only can provide security when  $\Phi$  is *false*; in particular, if  $\Phi$  is true, even if it is computationally hard to find a witness for

$\Phi$ , no guarantees are made about the encrypted message being hidden. Thus, it seems like we would need a *statistically-sound* ZK argument, i.e., a ZK proof: if the verifier’s first-round message is honest, with high probability, there should not exist an accepting second-round ZK message.

It is well-known that to achieve ZK in two rounds, it is necessary to have a simulator that runs in super-polynomial time (i.e., an SPS simulator). In every such known two-round ZK, the simulator works by brute-forcing some trapdoor provided in round 1, and giving proof that “either the statement is true or I found the trapdoor.” Because of the existence of this trapdoor, it would be impossible to make any such ZK argument statistically sound: an unbounded-time machine can always find the trapdoor and prove false statements. So it seems like the ZK scheme needs to satisfy two contradictory requirements: be statistically sound, and be a two-round scheme (which appears to preclude statistical soundness).

**Challenge 2: Non-malleability attacks.** Since the security of the underlying semi-malicious MrNISC holds only if the adversary knows some randomness for its messages, we need all parties to prove that they know the input and randomness corresponding to their messages. We are aiming for a protocol that can be evaluated in two rounds, so this necessitates using a non-malleable commitment (to prevent an attacker from, say copying the round 1 message of some other party). Unfortunately, non-interactive non-malleable commitments without setup are only known from very strong non-standard assumptions, such as adaptive one-way functions [39], hardness amplifiability [1, 18], and/or keyless hash functions [17, 18, 38]. These are very strong and non-standard assumptions, for some of which we have no plain-model instantiation, except heuristic ones. Thus, we want to achieve a secure MrNISC protocol (in the plain model) without such strong assumptions.

**Challenge 3: Adaptive reusability of the primitives.** We emphasize that we are building an MrNISC protocol, which significantly strengthens standalone two-round MPC. Because of this, our ZK argument and commitment schemes must satisfy strong forms of reusability. There are several challenges in ensuring both the ZK argument and non-malleable commitment scheme satisfy the types of reusability that we need, and we introduce several new ideas to solve these challenges. We will elaborate on this challenge below after we describe our ideas for solving challenges 1 and 2.

### Solving Challenge 1: How Do We Get a “statistically-Sound” SPS ZK?

We now discuss how to achieve the seemingly contradictory requirements of getting a 2-round SPS ZK argument which has a statistical soundness property that would allow it to be a witness for the WE scheme. Our key idea is to relax the notion of statistical soundness to one that is obtainable in two rounds but still sufficient to use with WE.

Imagine we have a WE scheme where the distinguishing advantage of an adversary is tiny (say, subexponential in  $\lambda$ ). It would then suffice to have a ZK

protocol that is statistically sound a negligible fraction of the time, as long as it is quite a bit larger than the distinguishing advantage of the WE. In more detail, consider a hypothetical zero-knowledge protocol with the following properties:

- The first round between a computationally-bounded verifier and a prover fully specifies one of the two possible “modes”: a *statistical ZK mode* and a *perfectly sound mode*.
- The perfectly sound mode occurs with some negligible probability  $\epsilon$ , and in this mode, no accepting round 2 message exists for any false statement
- In the statistical ZK mode (which occurs with overwhelming probability  $1 - \epsilon$ ), the second message is simulatable by an SPS machine and a simulated transcript is statistically indistinguishable from a normal transcript.
- Furthermore, it is computationally difficult for a malicious prover to distinguish between the two modes.

If we had such a ZK protocol, it would enable us to argue hiding of the witness encryption scheme whenever the first round of the protocol is not honest. The idea of this argument is as follows. Suppose an adversary could learn something about the second-round messages from their witness encryptions in some world where the first round was not honest. In that case, it should also be able to do so even in the perfectly-sound mode (otherwise, it would distinguish the modes). But in this mode, proofs for false statements do not exist; thus, the witness encryption provides full security. Even though this mode happens with negligible probability, it is still enough to contradict witness encryption security, whose advantage is much smaller.

To construct this new ZK scheme, we use ideas that are inspired by the extractable commitment scheme of Kalai, Khurana, and Sahai [33]. This commitment scheme has the property that it is extractable with some negligible tunable probability but is also statistically hiding. This commitment was used in the works of [6] to get a two-round statistical zero-knowledge argument with super-polynomial simulation. To instantiate our new “sometimes perfectly-sound” ZK argument, we use the protocol of [6] as a starting point, but we will need to make significant modifications. Namely, to force a well-defined perfect soundness mode, we will make the first round of this protocol a “simultaneous-message” round, where both the prover and the verifier send a message. We elaborate further on this and other key ideas used in our construction in the full version of the paper [26].

We note an important subtlety in this new definition and our construction. Namely, the statistical ZK and perfect soundness properties only hold with respect to the *second* round. If the verifier is unbounded-time, then after seeing a first-round prover’s message, it can send a first-round verifier’s message that forces perfect soundness all the time and thus disallows any prover from giving a simulated proof. On the other hand, if the prover is unbounded-time, then after seeing a first-round verifier’s message, it can send a first-round prover’s message, which causes the probability  $\epsilon$  of the perfect soundness mode to be 0. Thus the frequency of perfect soundness mode and the ability of the simulator to give a

simulated proof depend on the first round being generated by computationally bounded machines.

## Solving Challenge 2: How Do We Avoid Non-interactive Non-malleability?

To solve challenge two, we must somehow get a non-malleable commitment (NMC) scheme which can be executed in the first round without using strong assumptions such as keyless hash functions, hardness amplifiability, or adaptive one-way functions. Recall that unfortunately, all known instantiations of non-interactive NMCs (for a super-polynomial number of tags) currently require the use of (some combination of) these strong assumptions, so it seems at first glance that avoiding them would require making substantial progress on the difficult and well-studied question of non-interactive NMCs.

Our approach to solving this problem is inspired by the exciting work of Khurana [36], which builds a new type of commitment that works as follows. The commitment phase is similar to a non-interactive commitment in that the only communication from the committer is a first-round message  $C$ . The role of the receiver is slightly different: The receiver chooses a random string  $\tau$  internally, and it is both  $C$  and  $\tau$  together that truly defines the commitment (and, correspondingly, the underlying value being committed to). Consequently, to compute an opening, the committer must receive a  $\tau$  from the receiver. Non-malleability (and binding) hinges upon the fact that the  $\tau$  chosen by the receiver is chosen after seeing the commitment. (See the left diagram below for an illustration of this scheme.) Crucially, this commitment can be constructed from well-founded assumptions (indistinguishability obfuscation, time-lock puzzles, and OWPs), bypassing the need for the strong assumptions discussed earlier (Fig. 1).



**Fig. 1.** The diagram on the left depicts the communication pattern of Khurana’s [36] commitment scheme, whereas the diagram on the right depicts ours. The key difference is that in our scheme, the receiver’s message and the sender’s messages can be sent simultaneously, while in [36] the receiver’s message must be sent after the sender’s message.

We would like to use this commitment scheme in our protocol. There are two main issues that arise.

- First, to use this scheme, we would need the commitment phase to happen entirely in the first round. Namely, the receiver must publish  $\tau$  simultaneously while the committer is publishing  $C$ . (See the right-hand diagram above.) In

- particular, in the security proof, we need to handle the case of malicious committers who publish  $C$  after seeing the round-1  $\tau$ .
- Second, our goal is to have every party use this commitment to commit to their input and randomness for the protocol. Recall that in the scheme of [36], a well-defined commitment  $(C_j, \tau_i)$  consists of *both* the committer’s message  $C_j$  and the receiver’s random string  $\tau_i$ . Although honest parties  $P_j$  will always provide commitments  $C_j$  which are consistent across all  $\tau_i$ , it is perfectly plausible for a corrupted party to publish some  $C_j$  where different  $\tau_i$  yield commitments  $(C_j, \tau_i)$  to different values.

Solving the first issue involves identifying some technical challenges in the security proof of [36] and making changes to the protocol to avoid these issues. Because of this issue, the non-malleable commitment of [36] is really a two round commitment scheme. In this paper, relying on the axis of hardness given by a time-lock puzzle that we additionally use as an assumption, we construct a truly one round non-malleable commitment scheme, in the simultaneous message model. For the second issue, we use a surprisingly simple idea of adding a standard (potentially malleable) perfectly binding commitment scheme (e.g., Blum’s commitment) at the MrNISC protocol level, we can use this NMC scheme even though it does not satisfy the standard notion of binding. A more detailed technical overview of the non-malleable commitment scheme, as well as the formal construction, can be found in the full version of the paper [26].

### Solving Challenge 3: How Do We Get Reusability?

We now describe the challenges which arise when trying to get the type of reusability required by MrNISC. The main problem is to ensure that all of the building blocks we use (i.e., the ZK scheme and the NMC scheme) support the reuse of their first-round message. It turns out that the non-malleable commitment we described in the previous section can be adapted to this reusable setting without much modification. However, several challenges arise when adapting the sometimes-statistically-sound ZK scheme, which we discussed earlier, to the reusable setting. We focus on these challenges here.

Recall that the ZK scheme is a simultaneous message protocol, so a transcript consists of three messages of the form  $(zk_{1,P}, zk_{1,V}, zk_{2,P})$ , a round-1 message of the prover and the verifier, and a round-2 message of the prover. What we need is for any prover to be able to publish a single  $zk_{1,P}$  in round 1, which can be used in many different sessions with respect to many different  $zk_{1,V}$  messages. In addition, we require a very strong form of reusability: even if a malicious verifier sees an entire transcript  $(zk_{1,P}, zk_{1,V}, zk_{2,P})$ , and then chooses a new verifier’s first-round message  $zk'_{1,V}$ , zero-knowledge should still hold when the prover publishes a proof with respect to  $zk'_{1,V}$  and the prover’s *original* message  $zk_{1,P}$ . Similarly, a verifier should be able to publish a single  $zk_{1,V}$  which can be used in many different sessions with respect to many different  $zk_{1,P}$  messages, and the soundness properties of the ZK scheme should still hold.

Note that it is not immediately clear whether this reusability for ZK arguments are implied by a corresponding non-reusable version of ZK arguments. This turns out not to be the case. To satisfy reusability, we end up having to make several changes to our (non-reusable) sometimes-perfectly-sound ZK scheme. We again describe this in more detail in the full version of the paper [26].

## Putting Things Together

We now have the main pieces that we will use to construct a malicious-secure MrNISC: the two-round sometimes-statistically-sound ZK, receiver-assisted one-round CCA-secure commitment, and the underlying semi-malicious MrNISC. Significant challenges arise when attempting to combine these pieces in the way described earlier to get a malicious MrNISC protocol. To see this, it will be convenient to briefly mention the approach we take for the security proof.

A simplified version of the sequence of hybrids we use is as follows. First, we extract the value underlying the commitments and check if anyone acted dishonestly. If so, we switch the honest parties' witness encryptions to encrypt 0 rather than the actual round 2 messages (this is hybrid 1). Second, we simulate the ZK proof (this is hybrid 2). Third, we switch the underlying value in the commitment to 0 (this is hybrid 3). Once the commitments are independent of the true input, we can use the simulator of the underlying MrNISC (this is hybrid 4). The last hybrid is identical to our simulator.

To make the transitions between the hybrids possible, we need to set the hardness of every primitive carefully. Each hybrid indistinguishability induces some hardness inequality for the involved primitives. Unfortunately, the inequalities seem to be in contradiction to each other. Observe that for the first indistinguishability (between hybrid 0 and hybrid 1), we need our ZK argument's soundness properties to hold against adversaries who can run the CCA extractor. That is,

$$T_{\text{extractor}} \ll T_{\text{sound}}.$$

For the transition between hybrid 2 to 3, we need to guarantee that the security of the commitment scheme holds even against an adversary that can run the ZK simulator. That is,

$$T_{\text{ZKSim}} \ll T_{\text{extractor}}.$$

Together, the above two inequalities imply that it is necessary to have  $T_{\text{ZKSim}} \ll T_{\text{sound}}$ . But this is impossible, at least using the techniques we use in constructing the ZK argument. Our simulator works by brute-forcing the verifier's  $\text{zk}_{1,V}$  message to obtain some secret and produces proofs with this knowledge. In other words, whoever has the secret can produce accepting proofs without knowing a witness—this is essentially an upper bound on the soundness of the scheme, i.e.,  $T_{\text{sound}} \ll T_{\text{ZKSim}}$ , which means that our inequalities cannot be satisfied at the same time.

To solve this problem, we introduce another axis of hardness, namely, *circuit depth*. In particular, assume that it is possible to run the ZK simulator in some super-polynomial depth  $d$ . To do this, we would have to construct a ZK argument



where the secret embedded in  $\mathbf{zk}_{1,V}$  is extractable in depth  $d$ . Further, assume that in polynomial depth, it is extremely hard to extract the secret from  $\mathbf{zk}_{1,V}$  (much harder than size  $d$ ). We can use such a ZK argument to solve the problem above. Namely, we can restrict the reduction for hybrids 0 and 1 to run in *polynomial depth*, and in this complexity class, it holds that  $T_{\text{extractor}} \ll T_{\text{sound}}$ . For the reduction for hybrids 2 and 3, we will allow the depth to be  $d$ , in which case the inequality  $T_{\text{ZKSim}} \ll T_{\text{extractor}}$  is satisfied.

So we have reduced this problem to constructing a ZK argument which is simulatable in some super-polynomial depth  $d$  and whose soundness holds against size much larger than  $d$  as long as the depth is restricted to be polynomial. It turns out that it is possible to modify our original ZK argument to satisfy this property. We describe how to do this in the full version [26].

Several more minor technical issues arise when putting things together. One such problem is that of “simulation soundness,” that is, we need to guarantee that the adversary cannot give valid ZK arguments for false statements even if it sees simulated arguments from the honest parties. We solve this issue using techniques from the work of [7]. At a very high level, if we use a ZK argument where the simulated proofs are indistinguishable from normal proofs even to an adversary who is powerful enough to run the simulator itself, and if we commit to the witnesses using a non-malleable commitment, it is possible to design a sequence of hybrids that guarantees simulation soundness.

This and other minor technical details result in a construction and sequence of hybrids that are slightly more involved than the simplified version presented in this overview. We refer the reader to Sect. 6 for details.

### 3 Preliminaries

For any distribution  $\mathcal{X}$ , we denote by  $x \leftarrow \mathcal{X}$  the process of sampling a value  $x$  from the distribution  $\mathcal{X}$ . For a set  $X$  we denote by  $x \leftarrow X$  the process of sampling  $x$  from the uniform distribution over  $X$ . For an integer  $n \in \mathbb{N}$  we denote by  $[n]$  the set  $\{1, \dots, n\}$ . A function  $\text{negl} : \mathbb{N} \rightarrow \mathbb{R}$  is negligible if for every constant  $c > 0$  there exists an integer  $N_c$  such that  $\text{negl}(\lambda) < \lambda^{-c}$  for all  $\lambda > N_c$ . Throughout, when we refer to polynomials in security parameter, we mean constant degree polynomials that take positive value on non-negative inputs. We denote by  $\text{poly}(\lambda)$  an arbitrary polynomial in  $\lambda$  satisfying the above requirements of non-negativity.

Throughout this paper, all machines are assumed to be non-uniform. We will use  $\lambda$  to denote the security. We will use PPT as an acronym for “probabilistic (non-uniform) polynomial-time”. In addition, we use the notation  $T_1 \ll T_2$  (or  $T_2 \gg T_1$ ) if for all polynomials  $p$ ,  $p(T_1) < T_2$  asymptotically.

The statistical distance between two distributions  $X$  and  $Y$  over a discrete domain  $\Omega$  is defined as  $\Delta(X, Y) = (1/2) \cdot \sum_{\omega \in \Omega} |\Pr[X = \omega] - \Pr[Y = \omega]|$ .

**( $\mathcal{C}, \epsilon$ )-indistinguishability.** By  $\mathcal{C}$  we denote an abstract class of adversaries, where each adversary  $\mathcal{A} \in \mathcal{C}$  grows in some complexity measure (i.e. size, depth,

etc.) based on the security parameter  $\lambda$ . Security definitions will always hold with respect to some class of adversaries which we will specify.

**Definition 3.1** ( $(\mathcal{C}, \epsilon)$ -Indistinguishability). *Let  $\epsilon : \mathbb{N} \rightarrow (0, 1)$  be a function. We say that two distribution ensembles  $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$  and  $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$  are  $(\mathcal{C}, \epsilon)$ -indistinguishable if for any adversary  $\mathcal{A} \in \mathcal{C}$ , for any polynomial  $\text{poly}$ , and any  $\lambda \in \mathbb{N}$ ,*

$$\left| \Pr_{x \leftarrow \mathcal{X}_\lambda} [\mathcal{A}(1^\lambda, x)] - \Pr_{y \leftarrow \mathcal{Y}_\lambda} [\mathcal{A}(1^\lambda, y)] \right| \leq \epsilon(\lambda).$$

We use the shorthand  $\mathcal{X} \approx_{(\mathcal{C}, \epsilon)} \mathcal{Y}$  to denote this. If  $\mathcal{A}$  is unbounded time then we say that  $\mathcal{Y}$  and  $\mathcal{X}$  are statistically indistinguishable and we write  $\mathcal{X} \approx_{(\infty, \epsilon)} \mathcal{Y}$ , or alternately  $\Delta(\mathcal{X}, \mathcal{Y}) \leq \epsilon$ . (This corresponds to the standard definition of statistical distance.)

## 4 MrNISC Syntax and Security

We define the syntax of MrNISC and formalize security notions for malicious adversaries as well as semi-malicious adversaries, following the general framework given by Benhamouda and Lin [14].

We assume all parties have access to a broadcast channel, which any party can transmit a message to all other parties. We consider protocols given in the form of three polynomial-time algorithms (**Encode**, **Eval**, **Output**), where **Encode** and **Eval** are probabilistic, and **Output** is deterministic, for which we define the syntax as follows:

- **Input Encoding phase:** each party  $P_i$  computes  $m_{i,1} \leftarrow \text{Encode}(1^\lambda, x_i; r_{i,1})$ , where  $x_i$  is  $P_i$ 's private input, and the output  $m_{i,1}$  is  $P_i$ 's round 1 message.
- **Function Evaluation phase:** any set of parties  $I$  can compute an arity- $|I|$  function  $f$  on their respective inputs as follows. Each party  $P_i$  for  $i \in I$  computes  $m_{i,2} \leftarrow \text{Eval}(f, x_i, r_{i,1}, I, \{m_{i,1}\}_{i \in I}; r_{i,2})$ , where  $f$  is the function to compute,  $x_i$  is  $P_i$ 's private input,  $r_{i,1}$  is the randomness which  $P_i$  used to generate its input encoding,  $\{m_{i,1}\}_{i \in I}$  are the input encodings of all parties in  $I$ , and the output  $m_{i,2}$  is  $P_i$ 's round 2 message.
- **Output phase:** Anyone can compute  $y \leftarrow \text{Output}(\{m_{i,1}, m_{i,2}\}_{i \in I})$ .

**Malicious Security.** We follow the standard real/ideal paradigm in the following definition. An MrNISC scheme is malicious-secure for every PPT adversary  $\mathcal{A}$  in the real world there exists an ideal-world adversary  $\mathcal{S}$  (the “simulator”) such that the outputs of the following two experiments  $\text{Expt}_{\mathcal{A}}^{\text{Real}}(\lambda)$  and  $\text{Expt}_{\mathcal{A}, \mathcal{S}}^{\text{Ideal}}(\lambda)$  are indistinguishable.

In the following, for ease of exposition, we assume that each party sends at most one computation encoding for any  $(f, I)$  pair, and that parties ignore any subsequent computation encodings.

**Real Experiment**  $\text{Expt}_{\mathcal{A}}^{\text{Real}}(\lambda, z)$ . The experiment initializes the adversary  $\mathcal{A}$  with security parameter  $1^\lambda$  and auxiliary input  $z$ . In addition, the experiment initializes an empty list `honest_outputs`.  $\mathcal{A}$  chooses the number of parties  $M$  and the set of honest parties  $H \subseteq [M]$ .  $\mathcal{A}$  then submits queries to the experiment in an arbitrary number of iterations until it terminates. In every iteration  $k$ , it can submit one query of one of the following four types.

- **CORRUPT INPUT ENCODING:** The adversary  $\mathcal{A}$  can corrupt a party  $i \notin H$  and send an arbitrary first message  $m_{i,1}^*$  on its behalf.
- **HONEST INPUT ENCODING:** The adversary  $\mathcal{A}$  can choose an input  $x_i$  for honest party  $i$  and ask a party  $i \in H$  to send its first message by running  $m_{i,1}^* \leftarrow \text{Encode}(1^\lambda, x_i; r_{i,1})$ , where  $r_{i,1}$  is freshly chosen randomness. This  $m_{i,1}^*$  is sent to the adversary.
- **HONEST COMPUTATION ENCODING:** The adversary  $\mathcal{A}$  can ask an honest party  $i \in H$  to evaluate a function  $f$  on the inputs of parties  $I$ . If all first messages of parties in  $I$  are already published, party  $i$  computes and publishes  $m_{i,2}^* \leftarrow \text{Eval}(f, x_i, I, r_{i,1}, \{m_{i,1}^*\}_{i \in I}; r_{i,2})$ . Otherwise, the party instead publishes  $\perp$ .
- **CORRUPT COMPUTATION ENCODING:** The adversary can send an arbitrary function evaluation encoding  $m_{i,2}^*$  to the honest parties on behalf of some corrupted party  $i \notin H$  with respect to some function  $f$  and set  $I$ . If all parties in  $I$  have sent their `Eval` messages for  $(f, I)$ , the experiment adds the honest parties' output  $(f, I, \text{Output}(\{m_{i,1}^*, m_{i,2}^*\}_{i \in I}))$  to the list `honest_outputs`.

The output of the real experiment is defined to be  $(\text{view}_{\mathcal{A}}, \tau, \text{honest\_outputs})$ , where  $\text{view}_{\mathcal{A}}$  is the output of  $\mathcal{A}$  at the end of the computation, i.e. an arbitrary function of its view,  $\tau$  is the transcript of queries sent by  $\mathcal{A}$  along with the experiment's responses, and `honest_outputs` is the list defined above.

**Ideal Experiment**  $\text{Expt}_{\mathcal{A}, \mathcal{S}}^{\text{Ideal}}(\lambda, z)$ . The ideal experiment initializes  $\mathcal{A}$  with security parameter  $1^\lambda$  and auxiliary input  $z$ . After  $\mathcal{A}$  chooses the number of parties  $M$  and the set  $H \subsetneq [M]$ , the experiment initializes  $\mathcal{S}$  with  $1^\lambda$ ,  $M$ , and  $H$ . In addition, the experiment initializes an empty list `honest_outputs`. Subsequently, the adversary can make the same queries as in the real world, which are handled as follows:

- **CORRUPT INPUT ENCODING:** When  $\mathcal{A}$  sends a first message  $m_{i,1}^*$  on behalf of some party  $i \notin H$ , the experiment forwards this encoding to  $\mathcal{S}$ , who responds with an extracted input  $x_i$ .  $\mathcal{S}$  also has the option to declare that  $P_i$ 's input is  $\perp$ , which means that  $\mathcal{S}$  was not able to extract an input from  $m_{i,1}^*$  (for example, if the adversary sends a bogus string as its message). The experiment then sends  $x_i$  (if it is not  $\perp$ ) to the ideal functionality to be used as the input for party  $i$ .
- **HONEST INPUT ENCODING:** When the adversary  $\mathcal{A}$  chooses honest input  $x_i$  and asks party  $i \in H$  to send its first message, the experiment sends  $x_i$  to the ideal functionality to be used as the input for party  $i$ . The experiment then sends the index  $i$  (but not  $x_i$ ) to the simulator  $\mathcal{S}$ , who generates a simulated honest input encoding  $\tilde{m}_{i,1}$ . This encoding is forwarded back to  $\mathcal{A}$ .

- **HONEST COMPUTATION ENCODING:** When the adversary  $\mathcal{A}$  asks an honest party  $i \in H$  for a function evaluation encoding with respect to function  $f$  and parties  $I$ , assuming all parties in  $I$  have published input encodings, the experiment forwards this request to  $\mathcal{S}$ . If this is the last honest computation encoding generated with respect to  $f$  and  $I$ , and all corrupted parties in  $j \in I \setminus H$  have sent first messages  $m_{j,1}^*$  from which non- $\perp$  inputs have been extracted, then the experiment queries the ideal functionality on  $(f, I)$  to obtain the output  $y$ , which it forwards to the simulator as well. The simulator must then generate a simulated function evaluation encoding  $\tilde{m}_{i,2}$  on behalf of party  $i$ , regardless of whether it receives  $y$  or not. This encoding is forwarded to  $\mathcal{A}$ .
- **CORRUPT COMPUTATION ENCODING:** When the adversary sends a function evaluation encoding  $m_{i,2}^*$  on behalf of some corrupted party corresponding to some  $(f, I)$ , the experiment forwards  $(f, I, i, m_{i,2}^*)$  to the simulator. If all parties have sent computation encodings, the simulator chooses whether to allow the honest parties to learn the output corresponding to  $(f, I)$ . If so, the experiment adds  $(f, I, y)$  to the list `honest_outputs`; otherwise, the experiment adds  $(f, I, \perp)$  to `honest_outputs`.

The output of the ideal experiment is defined to be  $(\widehat{\text{view}}, \tau, \text{honest\_outputs})$ , where  $\widehat{\text{view}}$  is the output of  $\mathcal{A}$  at the end of the experiment,  $\tau$  is the transcript of queries made by  $\mathcal{A}$  along with the experiment's responses, and `honest_outputs` is the list defined above. In addition, at any point in the experiment,  $\mathcal{S}$  may choose to abort; in this case, the output of the experiment is whatever  $\mathcal{S}$  outputs at that point.

**Definition 4.1** ( $(\mathcal{C}_{\text{adv}}, \mathcal{C}_{\text{sim}}, \epsilon)$ -**Maliciously Secure MrNISC**). *We say that an MrNISC protocol  $\Pi$  is  $(\mathcal{C}_{\text{adv}}, \mathcal{C}_{\text{sim}}, \epsilon)$ -maliciously secure if for every  $\mathcal{C}_{\text{adv}}$  adversary  $(\mathcal{A}, \mathcal{D})$  there exists a  $\mathcal{C}_{\text{sim}}$  ideal-world adversary  $\mathcal{S}$  (i.e., the simulator) such that for every string  $z$ ,*

$$\left| \Pr[\mathcal{D}(\text{Expt}_{\mathcal{A}}^{\text{Real}}(\lambda, z)) = 1] - \Pr[\mathcal{D}(\text{Expt}_{\mathcal{A}, \mathcal{S}}^{\text{Ideal}}(\lambda, z)) = 1] \right| < \epsilon(\lambda).$$

The standard notion of security requires for every polynomial  $p(\cdot)$  the existence of a polynomial  $q(\cdot)$  for which the protocol is  $(p, q, \epsilon)$ -maliciously secure, where  $\epsilon(\cdot)$  is a negligible function. However, since we are interested in two-round protocols, it is known that the standard polynomial notion of security is impossible. Therefore, we focus on the relaxed notion of super-polynomial security (SPS): there is a sub-exponential function  $q(\cdot)$  such that for all polynomials  $p(\cdot)$ , the protocol is  $(p, q, \epsilon)$ -maliciously secure.

**The Semi-malicious Case.** We define a variant of the above security definition, which closely mirrors the definition of semi-malicious secure multiparty computation [5]. A *semi-malicious MrNISC adversary* is modeled as an algorithm which, whenever it sends a corrupted input or computation encoding on behalf of some party  $P_j$ , must also output some pair  $(x, r)$  which *explains its behavior*. More

specifically, all of the protocol messages sent by the adversary on behalf of  $P_j$  up to that point, including the message just sent, must exactly match the honest protocol specification for  $P_j$  when executed with input  $x$  and randomness  $r$ . Note that the witnesses given in different rounds need not be consistent. We also allow the adversary to “abort” a function evaluation in two different scenarios. First, instead of sending a CORRUPT INPUT ENCODING message for  $P_j$ , the adversary can send  $(j, \perp)$  to the experiment. In this case, the experiment will respond with  $\perp$  for all HONEST COMPUTATION ENCODING requests for  $(f, I)$ , and when all parties in  $I$  have been queried, it will add  $(f, I, \perp)$  to `honest_outputs`. Second, instead of sending a CORRUPT COMPUTATION ENCODING message on behalf of  $P_j$  the adversary can again send  $(j, f, I, \perp)$ . Again, after receiving such a query, the experiment will respond with  $\perp$  for all HONEST COMPUTATION ENCODING requests for  $(f, I)$ , and when all parties in  $I$  have been queried, it will add  $(f, I, \perp)$  to `honest_outputs`.

$I$  have published computation encodings for  $(f, I)$ . In this sense, the adversary may abort any individual function evaluation. Whenever an adversary aborts a CORRUPT INPUT ENCODING message on behalf of party  $P_j$ , it must abort any subsequent CORRUPT COMPUTATION ENCODING messages for  $P_j$ .

**Definition 4.2** ( $(\mathcal{C}_{\text{adv}}, \mathcal{C}_{\text{sim}}, \epsilon)$ -Semi-Malicious Secure MrNISC). *We say that an MrNISC protocol  $\Pi$  is  $(\mathcal{C}_{\text{adv}}, \mathcal{C}_{\text{sim}}, \epsilon)$ -semi-malicious secure if for every  $\mathcal{C}_{\text{adv}}$  semi-malicious adversary  $(\mathcal{A}, \mathcal{D})$  there exists  $\mathcal{C}_{\text{sim}}$  ideal-world adversary  $\mathcal{S}$  (i.e., the simulator) such that for every string  $z$ ,*

$$\left| \Pr[\mathcal{D}(\text{Expt}_{\mathcal{A}}^{\text{Real}}(\lambda, z)) = 1] - \Pr[\mathcal{D}(\text{Expt}_{\mathcal{A}, \mathcal{S}}^{\text{Ideal}}(\lambda, z)) = 1] \right| < \epsilon(\lambda).$$

## 5 Main Building Blocks

In this section, we give formal definitions for our new notion of reusable sometimes-statistically-sound zero-knowledge arguments along with the receiver-assisted one-round CCA-secure commitments, both of which we make use of in our MrNISC protocol.

### 5.1 Reusable Statistical ZK Arguments with Sometimes-Statistical Soundness

We define statistical zero-knowledge arguments with a specific communication pattern. The protocol that we need has a “simultaneous message” first round, where both the prover and verifier will simultaneously send a message. The syntax is the following:

1. The (honest) prover  $P = (\text{ZKProve}_1, \text{ZKProve}_2)$  and verifier  $V = (\text{ZKVerify}_1, \text{ZKVerify}_2)$  are each composed of two uniform PPT algorithms.
2.  $\text{ZKProve}_1$  and  $\text{ZKVerify}_1$  get as input only the security parameter  $\lambda$ .  $\text{ZKProve}_1$  outputs a message  $\text{zk}_{1,P}$  and a state  $\sigma_P$ .  $\text{ZKVerify}_1$  outputs a message  $\text{zk}_{1,V}$  and a state  $\sigma_V$ . The first round transcript is denoted  $\tau_1 = (\text{zk}_{1,P}, \text{zk}_{1,V})$ .

3. ZKProve<sub>2</sub> gets  $\sigma_P$ ,  $\text{zk}_{1,V}$ , the instance  $x$ , and a witness  $w$ . It outputs a message  $\text{zk}_{2,P}$ .
4. ZKVerify<sub>2</sub> gets the instance  $x$  and  $\tau = (\tau_1, \text{zk}_{2,P})$ , and outputs 0/1.

Looking ahead, we shall consider two-round ZK protocols as above with super-polynomial simulation (SPS), i.e., the simulator can run longer than the soundness bound. Further, we will also require that for a given prover and a verifier, the first message is reusable for proving multiple statements. We denote  $\langle P(w), V \rangle(1^\lambda, x)$  the output of the interaction between  $P$  and  $V$ , where  $P$  gets as input the witness  $w$ , and both  $P$  and  $V$  receive the instance  $x$  as a common input.

**Definition 5.1 (Reusable Statistical Zero-Knowledge Arguments with Sometimes-Statistical Soundness).** *Let  $L$  be a language in NP with a polynomial-time computable relation  $R_L$ . A protocol between  $P$  and  $V$  is a  $(\mathcal{C}_{\text{sound}}, \mathcal{C}_S, \mathcal{C}_{\text{zk}}, \epsilon_{\text{sound},1}, \epsilon_{\text{sound},2}, \epsilon_S)$ -reusable statistical zero-knowledge argument with sometimes-statistical soundness if it satisfies Definitions 5.2 to 5.4 below.*

**Definition 5.2 (Perfect Completeness).** *Let  $L$  be a language in NP with a polynomial-time computable relation  $R_L$ . A protocol between  $P$  and  $V$  satisfies perfect completeness if for every security parameter  $1^\lambda$  and  $(x, w) \in R_L$ , it holds that  $\Pr [\langle P(w), V \rangle(1^\lambda, x) = 1] = 1$ ,*

*where the probability is over the random coins of  $P$  and  $V$ .*

Additionally, we need a refined soundness property, defined next.

**Definition 5.3 ( $(\mathcal{C}_{\text{sound}}, \epsilon_{\text{sound},1}, \epsilon_{\text{sound},2})$ -statistical soundness).** *Consider any prover  $P^* \in \mathcal{C}_{\text{sound}}$  and a polynomial  $p(\cdot)$ , where on input the security parameter  $1^\lambda$ ,  $P^*$  outputs an instance  $x \in \{0, 1\}^p \setminus L$ . We require that there exists a “soundness mode indicator” machine  $\mathcal{E}$  that on input  $(\tau_1, \text{state}_V)$  outputs either 0 or 1 such that the following properties hold.*

- **Frequency of Soundness Mode.** *For every prover  $P^* \in \mathcal{C}_{\text{sound}}$ ,  $\Pr [\mathcal{E}(\tau_1, \text{state}_V) = 1] \geq \epsilon_{\text{sound},1}(\lambda)$ , where the probability is over the coins of the prover and the verifier in round 1.*
- **Perfect Soundness Holds During Soundness Mode.** *For every prover  $P^* \in \mathcal{C}_{\text{sound}}$  and every round-1 state  $(\tau_1, \text{state}_{P^*}, \text{state}_V)$  of the protocol, if  $\mathcal{E}(\tau_1, \text{state}_V) = 1$  then for all second-round messages  $\text{zk}_{2,P}$  sent by the prover corresponding to some false statement  $x \notin L$ , the verifier rejects on input  $(x, \tau_1, \text{zk}_{2,P}, \text{state}_V)$ .*
- **Indistinguishability of Soundness Mode.** *For every prover  $P^* \in \mathcal{C}_{\text{sound}}$ , it holds that*

$$\begin{aligned} & \{(\tau_1, \text{state}_{P^*}) \mid \mathcal{E}(\tau_1, \text{state}_V) = 1\} \\ & \quad \approx_{(\mathcal{C}_{\text{sound}}, \epsilon_{\text{sound},2})} \\ & \{(\tau_1, \text{state}_{P^*}) \mid \mathcal{E}(\tau_1, \text{state}_V) = 0\}. \end{aligned}$$

The full MrNISC protocol needs a powerful version of zero knowledge, as follows:

**Definition 5.4 (( $\mathcal{C}_S, \mathcal{C}_{zk}, \epsilon_S$ )-Adaptive Reusable Statistical Zero-Knowledge).** We say a zero knowledge scheme satisfies ( $\mathcal{C}_S, \mathcal{C}_{zk}, \epsilon_{S,1}, \epsilon_{S,2}$ )-adaptive reusable statistical zero-Knowledge if there exists a (uniform) simulator  $ZKSim \in \mathcal{C}_S$  which takes as input the round-one transcript  $\tau_1$ , the honest prover's state  $\sigma_P$ , and a statement  $x$  such that the following holds. Consider an adversary  $V^* \in \mathcal{C}_{zk}$  that takes as input  $1^\lambda$  and an honestly generated prover's first round message  $zk_{1,P}$ , and plays the following game  $\text{expt}_{V^*,zk}^b$ :

1.  $V^*$  may adaptively issue queries of the form  $(x, w, zk_{1,V}^*)$ . The challenger responds as follows:
  - $f(x, w) \notin R_L$ , the challenger responds with  $\perp$ .
  - If  $(x, w) \in R_L$  and  $b = 0$ , the challenger responds with the honest prover's second message  $ZKProve_2(\sigma_p, zk_{1,V}^*, x, w)$ .
  - If  $(x, w) \in R_L$  and  $b = 1$ , the challenger responds with the simulated prover's message  $ZKSim(\sigma_p, zk_{1,V}^*, x)$ .
2. At the end of the game,  $V^*$  outputs an arbitrary function of its view, which is used as the output of the experiment.

It must hold that  $\text{expt}_{V^*,zk}^0 \approx_{(\infty, \epsilon_S)} \text{expt}_{V^*,zk}^1$ .

An overview and complete details of our construction of the reusable SZK argument with sometimes-statistical soundness can be found in the full version of the paper [26].

## 5.2 One-Round Simultaneous-Message CCA-Non-malleable Commitments

In the following, we define the syntax and required security properties of the commitment scheme which we use in the main MrNISC construction in Sect. 6. This commitment is a *simultaneous-message one-round commitment*, where both committer and receiver send a message during the single round. The receiver's message is a uniform random string  $\tau$ , and the committer's message is some obfuscated program  $P$ . The committed value is only fixed when both  $P$  and  $\tau$  are fixed. To reflect this, in the definition of syntax below, `ComputeOpening`, `VerifyOpening`, and `CCAVal` take both the committer's message  $P$  and the receiver's message  $\tau$  as input.

Let  $\mathcal{T} = \{\mathcal{T}_\lambda\}_{\lambda \in \mathbb{N}}$  be the tag space which is  $[T(\lambda)]$ , where  $T = 2^{\text{poly}(\lambda)}$ . The modified syntax is as follows.

**Definition 5.5 (Syntax of one-round simultaneous-message CCA-non-malleable commitments).** With respect to the tag space  $\mathcal{T}$ , the NMC consists of the following algorithms.

$\text{CCACommit}(1^\lambda, \text{tag}, m; r)$  : The probabilistic polynomial time commitment algorithm takes as input the security parameter  $\lambda$ , a tag  $\text{tag} \in \mathcal{T}_\lambda$ , and a message  $m \in \{0, 1\}^*$ , and outputs a commitment  $P$ .

$\text{ComputeOpening}(\tau, \text{tag}, P, m, r)$  : The polynomial time deterministic algorithm

$\text{ComputeOpening}$  takes as input a string  $\tau \in \{0, 1\}^{\ell_t}$ , a tag  $\text{tag} \in \mathcal{T}_\lambda$ , a commitment  $P$ , a message  $m \in \{0, 1\}^*$ , and the randomness  $r$  used to commit. It outputs an opening  $\sigma \in \{0, 1\}^*$ . Above  $\ell_t = \ell_t(\lambda, n)$  is a polynomial associated with the scheme.

$\text{VerifyOpening}(\tau, \text{tag}, P, m, \sigma)$  : The polynomial-time deterministic algorithm  $\text{VerifyOpening}$  takes a string  $\tau \in \{0, 1\}^{\ell_t}$ , a tag  $\text{tag} \in \mathcal{T}_\lambda$ , a commitment  $P$ , a message  $m \in \{0, 1\}^*$ , and an opening  $\sigma$ . It outputs a value in  $\{0, 1\}$ .

Such a scheme is said to be a one-round simultaneous-message CCA-non-malleable commitment if it satisfies the following properties:

**Definition 5.6 (Correctness of Opening).** Let  $\lambda \in \mathbb{N}$  be the security parameter, and consider any  $\text{tag} \in \mathcal{T}_\lambda$ , any message  $m \in \{0, 1\}^*$ , any  $\tau \in \{0, 1\}^{\ell_t}$ , any  $P \leftarrow \text{CCACommit}(1^\lambda, \text{tag}, m; r)$ . Then,  $\Pr[\text{VerifyOpening}(\tau, \text{tag}, P, m, \sigma) = 1] = 1$ , where  $\sigma = \text{ComputeOpening}(\tau, \text{tag}, P, m, r)$ .

**Definition 5.7 (Extraction).** There exists an (inefficient) algorithm  $\text{CCAVal}$  with the following properties. For any  $\lambda \in \mathbb{N}$  and any message  $m \in \{0, 1\}^*$ , tag  $\text{tag} \in \mathcal{T}_\lambda$ , commitment  $P$ , and  $\tau \in \{0, 1\}^{\ell_t(\lambda)}$ , it holds that

$$\left( \exists \sigma : \text{VerifyOpening}(\tau, \text{tag}, P, m, \sigma) = 1 \right) \iff \text{CCAVal}(\tau, \text{tag}, P) = m.$$

In addition,  $\text{CCAVal}$  runs in time  $2^{\text{poly}(\lambda)}$  for some fixed polynomial  $\text{poly}$ .

We now specify the CCA security property.

**Definition 5.8 (( $\mathcal{C}, \epsilon$ )-CCA security).** We define the following security game played between the adversary  $\mathcal{A} \in \mathcal{C}$  and the challenger. We denote it by  $\text{expt}_{\mathcal{A}, \text{CCA}}(1^\lambda)$ :

1. The challenger manages a list  $L$  that is initially empty. The contents of the list are visible to the adversary at all stages.
2. The adversary sends a challenge tag  $\text{tag}^* \in \mathcal{T}_\lambda$ .
3. The adversary submits queries of the following kind in an adaptive manner:
  - (a) Adversary can ask for arbitrary polynomially many  $\tau$ -queries. Challenger samples  $\tau' \leftarrow \{0, 1\}^{\ell_t}$  and appends  $\tau'$  to  $L$ .
  - (b) Adversary can ask for an arbitrary polynomially many  $(\tau, \text{tag}, P)$ -queries for any  $\tau \in L$ , any  $\text{tag} \neq \text{tag}^*$ , and any commitment  $P$ . The challenger computes  $\text{CCAVal}(\tau, \text{tag}, P)$  and sends the result to the adversary.
4. The adversary submits two messages  $m_0, m_1 \in \mathcal{M}_\lambda$ . The challenger samples  $b \leftarrow \{0, 1\}$ , and computes  $P^* \leftarrow \text{CCACommit}(1^\lambda, \text{tag}^*, m_b)$ . The adversary gets  $P^*$  from the challenger.



5. *The adversary repeats Step 3.*
6. *Finally, the adversary outputs a guess  $b' \in \{0, 1\}$ . The experiment outputs 1 if  $b' = b$  and 0 otherwise.*

*The one-round (simultaneous-message) CCA-secure commitment scheme CCA scheme satisfies  $(\mathcal{C}, \epsilon)$ -CCA security if for all adversaries  $\mathcal{A} \in \mathcal{C}$ :*

$$\left| \Pr[\text{expt}_{\mathcal{A}, \text{CCA}}(1^\lambda) = 1] - \frac{1}{2} \right| \leq \epsilon.$$

Our NMC construction is an extension of [36]. It takes the same form as that of [36], namely, the committer publishes a message  $P$ , and the receiver publishes a random  $\tau$ . We change the internals of the construction, though, to allow the receiver to publish  $\tau$  during the first round, simultaneously while the committer is publishing  $P$ . We show that with our modifications, even a rushing committer who chooses  $P$  based on  $\tau$  cannot break security. Thus we achieve a (simultaneous-message) one-round NMC which satisfies the full CCA security definition given above, relying on iO and other standard assumptions. We refer to the full version of the paper [26] for details.

## 6 Malicious-Secure MrNISC

In this section, we give the formal construction and proof of security for our MrNISC protocol.

**Required Primitives and Parameters.** We make use of the following primitives in our construction.

- *Commitment:* A non-interactive perfectly binding commitment (NICommit).
- *Pseudo-Random Function* A pseudo-random function (*PRF*).
- *Witness Encryption:* We use witness encryption We use circuit SAT as our NP language.
- *Reusable Statistical ZK Arguments with Sometimes-Statistical Soundness:* We use a SPS ZK argument ( $\text{ZKProve}_1, \text{ZKVerify}_1, \text{ZKProve}_2, \text{ZKVerify}_2$ ) satisfying Definitions 5.1, 5.3 and 5.4.
- *One-round CCA commitments:* We use one-round (simultaneous-message) CCA commitments as in Definitions 5.5 to 5.8.
- *Semi-malicious MrNISC:* We use an underlying semi-malicious MrNISC protocol ( $\text{SM.Encode}, \text{SM.Eval}, \text{SM.Output}$ ), satisfying the security notion given in Definition 4.2.

**Complexity Hierarchy.** In order to argue security, we require that the primitives we use are secure against adversaries of varying complexities. In particular, we require the following complexity hierarchy to hold with respect to the primitives. Let  $T_1, T_2, T_3, T_4, T_5$  be functions over  $\lambda$ , such that

$$T_1 \ll T_2 \ll T_3 \ll T_4 \ll T_5,$$

where  $T \ll T'$  means that  $p(T) < T'$  asymptotically for all polynomials  $p$ . We require the following:

- The ZK argument scheme satisfies  $(\mathcal{C}_S, \mathcal{C}_{zk}, \epsilon_S)$ -adaptive reusable statistical zero knowledge (Definition 5.4) where  $\mathcal{C}_S$  is the class of circuits of size  $\text{poly}(T_1)$  and depth  $T_1$  (i.e. the simulator runs in size  $\text{poly}(T_1)$  and depth  $T_1$ ), and  $\mathcal{C}_{zk}$  is the class of circuits of size  $p(T_3)$  for all polynomials  $p$ , and  $\epsilon_S$  is any negligible function (i.e. statistical zero knowledge holds as long as the verifier's first-round message is generated by a machine in  $\mathcal{C}_{zk}$ ).
- The CCA non-malleable commitment scheme satisfies  $(\mathcal{C}, \epsilon)$ -CCA security, where  $\mathcal{C}$  is the class of circuits of size  $p(T_1)$  for all polynomials  $p$ , and  $\epsilon$  is any negligible function.
- The CCA non-malleable commitment scheme's extractor  $\text{CCAVal}$  is a circuit of size  $T_2$  and polynomial depth.
- The perfectly-binding commitment scheme is hiding against adversaries of size  $p(T_2)$  for all polynomials  $p$ , and is extractable by a circuit of size  $T_3$ .
- The ZK argument scheme satisfies  $(\mathcal{C}_{\text{sound}}, \epsilon_{\text{sound},1}, \epsilon_{\text{sound},2})$ -statistical soundness, where  $\mathcal{C}_{\text{sound}}$  is the class of circuits of size  $p(T_5)$  and polynomial depth for all polynomials  $p$  (refer to Definition 5.3 for details on the meaning of  $\mathcal{C}_{\text{sound}}$ ), and  $\epsilon_{\text{sound},1} = 1/T_4$ , and  $\epsilon_{\text{sound},2}$  is any negligible function.
- The witness encryption scheme satisfies  $(\mathcal{C}, \epsilon)$ -security, where  $\mathcal{C}$  is the class of circuits of size  $p(T_5)$  for all polynomials  $p$ , and  $\epsilon = 1/T_5$ .
- The pseudo-random function is secure against adversaries of size  $p(T_5)$  for all polynomials  $p$ .
- The semi-malicious MrNISC protocol is secure against adversaries of size  $p(T_5)$  for all polynomials  $p$ .

### The Relation $\Phi_{zk,i,j}$

**Hardwired:** The function  $f$  and the set  $I$ ,  $P_i$ 's tag  $\text{tag}_i$ ,  $P_i$ 's CCA non-malleable commitment  $\text{nmc}_i$ ,  $P_i$ 's perfectly binding commitment  $\text{com}_i$ ,  $P_i$ 's first round semi-malicious MPC message  $\hat{m}_{i,1}$ ,  $P_j$ 's string  $\tau_j$ ,  $P_i$ 's commitment  $\text{com}_{i,\hat{m}_{i,2}}$  to its semi-malicious evaluation encoding  $\hat{m}_{i,2}$ , and the transcript  $\rho_{\text{sm},1}$  of the semi-malicious input encodings of all parties from  $I$ .

**Input/Witness:**  $W_{zk,i} = (x_i, r_{i,\text{SM},1}, K_i, r_{i,\text{com}}, \sigma_{i,j,\text{CCA}}, \hat{m}_{i,2})$ .

**Computation:** Verify the following steps.

1.  $\text{VerifyOpening}(\tau_j, \text{tag}_i, \text{nmc}_i, (x_i, r_{i,\text{SM},1}, K_i, r_{i,\text{com}}), \sigma_{i,j,\text{CCA}}) = 1$
2.  $\text{com}_i = \text{NICommit}(1^\lambda, (x_i, r_{i,\text{SM},1}, K_i); r_{i,\text{com}})$
3.  $\hat{m}_{i,1} = \text{SM.Encode}(1^\lambda, x_i, r_{i,\text{SM},1})$
4.  $\hat{m}_{i,2} = \text{SM.Eval}(f, x_i, r_{i,\text{SM},1}, I, \rho_{\text{sm},1}; \text{PRF}_{K_i}(f, I, 1))$
5.  $\text{com}_{i,\hat{m}_{i,2}} = \text{NICommit}(1^\lambda, \hat{m}_{i,2}; \text{PRF}_{K_i}(f, I, 2))$

Output 1 if all the above checks succeed, otherwise output 0.

**The Relation  $\Phi_{WE,i}$**

**Hardwired:** The function  $f$ , the set  $I$ , the set of tags of all parties,  $P_i$ 's first-round verifier zk message  $zk_{1,i,V}$ ,  $P_i$ 's string  $\tau_i$ , the first-round prover zk messages, commitments and semi-malicious encodings  $\{zk_{1,j,P}, \hat{m}_{j,1}, com_j, nmc_j\}_{j \in I \setminus \{i\}}$  included in the input encodings of all other parties in  $I$ .

**Witness:**

$$W_{WE,i} = (\{zk_{2,j \rightarrow i,P}, com_{j,\hat{m}_{j,2}}\}_{j \neq i}).$$

**Computation:** For every  $j \in I \setminus \{i\}$ ,

1. Let

$$\Phi_{zk,j} = \Phi_{zk,j}[f, I, tag_j, nmc_j, com_j, \hat{m}_{j,1}, \tau_i, com_{j,\hat{m}_{j,2}}, \rho_{sm,1}]$$

be the circuit described in page 122, with the values

$$[f, I, tag_j, nmc_j, com_j, \hat{m}_{j,1}, \tau_i, com_{j,\hat{m}_{j,2}}, \rho_{sm,1}]$$

hardcoded.

2. Compute  $ZKVerify_2(\Phi_{zk,j}, zk_{1,i,V}, zk_{1,j,P}, zk_{2,j \rightarrow i,P})$ .

Output 1 if all the above checks succeed, otherwise output 0.

**Protocol.** We give the protocol below, described in terms of the behavior of party  $P_i$  during the input encoding phase, the evaluation phase, and the output computation phase. In particular, we give this behavior by implementing the Encode, Eval and Output algorithms defined in Section 4. Assume that each party  $P_i$  has input  $x_i$  and a public identity denoted by  $tag_i \in \mathcal{T}_\lambda$ . Note that the Output algorithm is public and can be performed without  $P_i$ 's private input or state. Throughout the protocol description, we deal with PPT algorithms as follows. If a PPT algorithm  $P$  is invoked on some input  $x$  without any randomness explicitly given (i.e., we write  $P(x)$ ), we implicitly assume that it is supplied with freshly chosen randomness. In some cases we will need to explicitly manipulate the randomness of algorithms, in which case we will write  $P(x; r)$ .

- **Input Encoding**  $Encode(1^\lambda, tag_i, x_i)$ : The input encoding algorithm takes as input  $1^\lambda$ , where  $\lambda$  is the security parameter, along with  $P_i$ 's tag  $tag_i$  and private input  $x_i$ , and does the following.

1. Compute the input encoding  $\hat{m}_{i,1} \leftarrow SM.Encode(1^\lambda, x_i; r_{i,SM,1})$  from the semi-malicious protocol, where  $r_{i,SM,1} \xleftarrow{\$} \{0, 1\}^*$  is freshly chosen randomness.
2. Choose a PRF key  $K_i$ .
3. Compute a perfectly binding commitment

$$com_i \leftarrow NICommit(1^\lambda, (x_i, r_{i,SM,1}, K_i); r_{i,com})$$

of the input and the semi-malicious encoding randomness, where  $r_{i,com} \xleftarrow{\$} \{0, 1\}^*$  is freshly chosen randomness.

4. Compute a CCA-non-malleable commitment

$$nmc_i \leftarrow CCACommit(1^\lambda, tag_i, (x_i, r_{i,SM}, K_i, r_{i,com}); r_{i,CCA})$$

of the same values committed to in the perfectly binding commitment, along with the randomness used for generating the perfectly binding commitment, where  $r_{i,CCA} \stackrel{\$}{\leftarrow} \{0,1\}^*$  is freshly chosen randomness.

5. Compute a random string  $\tau_i \stackrel{\$}{\leftarrow} \{0,1\}^\ell$ .
6. Compute the first round verifier's message and state

$$(\sigma_{zk,1,i,V}, \mathbf{zk}_{1,i,V}) \leftarrow \text{ZKVerify}_1(1^\lambda)$$

and the first round prover message and state

$$(\sigma_{zk,1,i,P}, \mathbf{zk}_{1,i,P}) \leftarrow \text{ZKProve}_1(1^\lambda).$$

7. Output  $m_{i,1} = (\hat{m}_{i,1}, \text{com}_i, \text{nmc}_i, \tau_i, \mathbf{zk}_{1,i,V}, \mathbf{zk}_{1,i,P})$ .

– **Function Evaluation**  $\text{Eval}(f, \text{tag}_i, x_i, r_{i,1}, I, \rho_1)$ : The function evaluation algorithm takes as input the function  $f$  to be evaluated, the set  $I$  of participating parties,  $P_i$ 's private input  $x_i$ , the randomness  $r_{i,1}$  which  $P_i$  used to generate its input encoding, and the input encoding transcript  $\rho_1$ , and does the following:

1. Parse  $\rho_1 = \{\hat{m}_{k,1}, \text{com}_k, \text{nmc}_k, \tau_k, \mathbf{zk}_{1,k,V}, \mathbf{zk}_{1,k,P}\}_{k \in [n]}$  to obtain  $(r_{i,SM,1}, r_{i,com}, r_{i,CCA}, \sigma_{zk,1,i,V}, \sigma_{zk,1,i,P})$  from  $r_{i,1}$ .
2. Compute the semi-malicious function evaluation encoding

$$\hat{m}_{i,2} \leftarrow \text{SM.Eval}(f, x_i, r_{i,SM,1}, I, \rho_{sm,1}; \text{PRF}_{K_i}(f, I, 1))$$

of the underlying semi-malicious protocol, using the transcript  $\rho_{sm,1} = \{\hat{m}_{k,1}\}_{k \in I}$  of the semi-malicious input encodings of all parties from  $I$ , where the randomness is chosen using the PRF key committed to during the input encoding phase.

3. Compute a commitment  $\text{com}_{i,\hat{m}_{i,2}} \leftarrow \text{NICommit}(\hat{m}_{i,2}; \text{PRF}_{K_i}(f, I, 2))$  of the encoding  $\hat{m}_{i,2}$  using randomness derived from the PRF key committed to during the input encoding phase.
4. For each  $P_j, j \in I \setminus \{i\}$ :
  - Compute an opening

$$\sigma_{i,j,CCA} \leftarrow \text{ComputeOpening}(\tau_j, \text{tag}_i, \text{nmc}_i, (x_i, r_{i,SM,1}, K_i, r_{i,com}), r_{i,CCA})$$

for the non-malleable-commitment  $\text{nmc}_i$  with respect to  $\tau_j$ .

- Compute
  - a round two ZK prover's message  $\mathbf{zk}_{2,i \rightarrow j,P} \leftarrow \text{ZKProve}_2(\Phi_{zk,i,j}, W_{zk,i}, \sigma_{zk,1,i,P}, \mathbf{zk}_{1,j,V})$ , where  $\Phi_{zk,i,j}$  is the circuit SAT instance defined on page 6. Here  $W_{zk,i} = (x_i, r_{i,SM,1}, K_i, r_{i,com}, \sigma_{i,j,CCA}, \hat{m}_{i,2})$  is the witness for generating this prover message.
5. Compute a witness encryption  $\text{WE}_i \leftarrow \text{WE.Encrypt}(1^\lambda, \Phi_{\text{WE},i}, r_{com,i,\hat{m}_{i,2}})$  where the circuit  $\Phi_{\text{WE},i}$  is described on page 26, and the plaintext  $r_{com,i,\hat{m}_{i,2}} = \text{PRF}_{K_i}(f, I, 2)$  is the opening for  $\text{com}_{i,\hat{m}_{i,2}}$ .
6. Return  $m_{i,2} = (\text{com}_{i,\hat{m}_{i,2}}, \{\mathbf{zk}_{2,i \rightarrow j,P}\}_{j \in I \setminus \{i\}}, \text{WE}_i)$ .

- **Output Computation**  $\text{Output}(\{m_{j,1}, m_{j,2}\}_{j \in I})$ : The output computation algorithm takes as input the input encoding  $m_{j,1}$  and the function evaluation encoding  $m_{j,2}$  of every party  $P_j$  for  $j \in I$  and does the following:

1. Parse

$$m_{j,1} = (\hat{m}_{j,1}, \text{com}_j, \text{nmc}_j, \tau_j, \text{zk}_{1,j,v}, \text{zk}_{1,j,p})$$

and

$$m_{j,2} = (\text{com}_{j,\hat{m}_{j,2}}, \{\text{zk}_{2,j \rightarrow k,P}\}_{k \in I \setminus \{j\}}, \text{WE}_j)$$

for each  $j \in I$ .

2. For each  $j, k \in I, j \neq k$ :
  - Run  $\text{ZKVerify}_2(\Phi_{\text{zk},j,k}, \text{zk}_{1,k,v}, \text{zk}_{1,j,p}, \text{zk}_{2,j \rightarrow k,p})$ , where  $\Phi_{\text{zk},j,k}$  is described on page 25. If the verification fails, abort and output  $\perp$ .
3. For each  $j \in I$ :
  - Compute the decryption  $r_{\text{com},j,\hat{m}_{j,2}} \leftarrow \text{WE.Decrypt}(\text{WE}_j, W_{\text{WE},j})$  of the opening  $r_{\text{com},j,\hat{m}_{j,2}}$  to the commitment  $\text{com}_{j,\hat{m}_{j,2}}$ , using the witness  $W_{\text{WE},j} = (\{\text{zk}_{2,k \rightarrow j,P}, \text{com}_{j,\hat{m}_{j,2}}\}_{k \neq j})$ . If the decryption fails, abort and output  $\perp$ .
  - Open  $\text{com}_{j,\hat{m}_{j,2}}$  to  $P_j$ 's semi-malicious function evaluation encoding  $\hat{m}_{j,2}$  using  $r_{\text{com},j,\hat{m}_{j,2}}$ .
4. Compute the output  $y \leftarrow \text{Output}(\{\hat{m}_{j,1}, \hat{m}_{j,2}\}_{j \in I})$  using the values  $\hat{m}_{j,2}$  obtained from decrypting the witness encryptions along with the semi-malicious input encodings  $\hat{m}_{j,2}$ .
5. Output  $y$ .

**Correctness.** Correctness of the protocol follows directly from correctness of the underlying primitives.

We refer to the full version [26] for the proof of security.

**Acknowledgement.** Rex Fernando is supported in part by a Simons Investigator Award, DARPA SIEVE award, NTT Research, NSF Frontier Award 1413955, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024. Ilan Komargodski is the incumbent of the Harry & Abe Sherman Senior Lectureship at the School of Computer Science and Engineering at the Hebrew University, supported in part by an Alon Young Faculty Fellowship, by a grant from the Israel Science Foundation (ISF Grant No. 1774/20), and by a grant from the US-Israel Binational Science Foundation and the US National Science Foundation (BSF-NSF Grant No. 2020643).

## References

1. Agarwal, A., Bartusek, J., Goyal, V., Khurana, D., Malavolta, G.: Two-round maliciously secure computation with super-polynomial simulation. In: TCC, pp. 654–685 (2021)

2. Ananth, P., Choudhuri, A.R., Jain, A.: A new approach to round-optimal secure multiparty computation. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 468–499. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63688-7\\_16](https://doi.org/10.1007/978-3-319-63688-7_16)
3. Ananth, P., Jain, A., Jin, Z., Malavolta, G.: Unbounded multi-party computation from learning with errors. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 754–781. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77886-6\\_26](https://doi.org/10.1007/978-3-030-77886-6_26)
4. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-29011-4\\_29](https://doi.org/10.1007/978-3-642-29011-4_29)
5. Asharov, G., Jain, A., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. Cryptology ePrint Archive, Report 2011/613 (2011). <https://eprint.iacr.org/2011/613>
6. Badrinarayanan, S., Fernando, R., Jain, A., Khurana, D., Sahai, A.: Statistical ZAP arguments. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 642–667. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45727-3\\_22](https://doi.org/10.1007/978-3-030-45727-3_22)
7. Badrinarayanan, S., Goyal, V., Jain, A., Khurana, D., Sahai, A.: Round optimal concurrent MPC via strong simulation. In: TCC, pp. 743–775 (2017)
8. Barak, B., Prabhakaran, M., Sahai, A.: Concurrent non-malleable zero knowledge. In: FOCS, pp. 345–354 (2006)
9. Bartusek, J., Garg, S., Masny, D., Mukherjee, P.: Reusable two-round MPC from DDH. In: TCC, pp. 320–348 (2020)
10. Bartusek, J., Garg, S., Srinivasan, A., Zhang, Y.: Reusable two-round MPC from LPN. In: PKC, pp. 165–193 (2022)
11. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: STOC, pp. 503–513. ACM Press (1990)
12. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC, pp. 1–10 (1988)
13. Benhamouda, F., Jain, A., Komargodski, I., Lin, H.: Multiparty reusable non-interactive secure computation from LWE. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 724–753. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77886-6\\_25](https://doi.org/10.1007/978-3-030-77886-6_25)
14. Benhamouda, F., Lin, H.: Mr NISC: multiparty reusable non-interactive secure computation. In: TCC, pp. 349–378 (2020)
15. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: STOC, pp. 111–120 (2013)
16. Bitansky, N., Kalai, Y.T., Paneth, O.: Multi-collision resistance: a paradigm for keyless hash functions. In: STOC, pp. 671–684, June 2018
17. Bitansky, N., Y., Kalai, T., Paneth, O.: Multi-collision resistance: a paradigm for keyless hash functions. In: STOC, pp. 671–684 (2018)
18. Bitansky, N., Lin, H.: One-message zero knowledge and non-malleable commitments. In: TCC, pp. 209–234 (2018)
19. Brakerski, Z., Halevi, S., Polychroniadou, A.: Four round secure computation without setup. In: TCC, pp. 645–677 (2017)

20. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions, pp. 541–550. IEEE Computer Society Press (2010)
21. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: STOC, pp. 11–19 (1988)
22. Choudhuri, A.R., Ciampi, M., Goyal, V., Jain, A., Ostrovsky, R.: Round optimal secure multiparty computation from minimal assumptions. In: TCC, pp. 291–319 (2020)
23. Ciampi, M., Ostrovsky, R., Siniscalchi, L., Visconti, I.: Round-optimal secure two-party computation from trapdoor permutations. In: TCC, pp. 678–710 (2017)
24. Dodis, Y., Jain, A., Moran, T., Wichs, D.: Counterexamples to hardness amplification beyond negligible. In: TCC, pp. 476–493 (2012)
25. Fernando, R., Gelles, Y., Komargodski, I., Shi, E.: Maliciously secure massively parallel computation for all-but-one corruptions. In: Dodis, Y., Shrimpton, T. (eds.) *Advances in Cryptology—CRYPTO 2022*. CRYPTO 2022. Lecture Notes in Computer Science, vol. 13507, pp. 688–718. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-15802-5\\_24](https://doi.org/10.1007/978-3-031-15802-5_24)
26. Fernando, R., Jain, A., Komargodski, I.: Maliciously-secure MrNISC in the plain model. *Cryptology ePrint Archive*, Report 2021/1319 (2021). <https://eprint.iacr.org/2021/1319>
27. Garg, R., Khurana, D., Lu, G., Waters, B.: Black-box non-interactive non-malleable commitments. In: Canteaut, A., Standaert, F.-X. (eds.) *EUROCRYPT 2021*. LNCS, vol. 12698, pp. 159–185. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77883-5\\_6](https://doi.org/10.1007/978-3-030-77883-5_6)
28. Garg, S., Goyal, V., Jain, A., Sahai, A.: Concurrently secure computation in constant rounds. In: Pointcheval, D., Johansson, T. (eds.) *EUROCRYPT 2012*. LNCS, vol. 7237, pp. 99–116. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-29011-4\\_8](https://doi.org/10.1007/978-3-642-29011-4_8)
29. Garg, S., Mukherjee, P., Pandey, O., Polychroniadou, A.: The exact round complexity of secure computation. In: Fischlin, M., Coron, J.-S. (eds.) *EUROCRYPT 2016*. LNCS, vol. 9666, pp. 448–476. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49896-5\\_16](https://doi.org/10.1007/978-3-662-49896-5_16)
30. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: STOC, pp. 218–229 (1987)
31. 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>
32. Kalai, Y.T., Khurana, D.: Non-interactive non-malleability from quantum supremacy. In: Boldyreva, A., Micciancio, D. (eds.) *CRYPTO 2019*. LNCS, vol. 11694, pp. 552–582. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-26954-8\\_18](https://doi.org/10.1007/978-3-030-26954-8_18)
33. Kalai, Y.T., Khurana, D., Sahai, A.: Statistical witness indistinguishability (and more) in two messages. In: Nielsen, J.B., Rijmen, V. (eds.) *EUROCRYPT 2018*. LNCS, vol. 10822, pp. 34–65. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-78372-7\\_2](https://doi.org/10.1007/978-3-319-78372-7_2)
34. Katz, J., Ostrovsky, R.: Round-optimal secure two-party computation. In: Franklin, M. (ed.) *CRYPTO 2004*. LNCS, vol. 3152, pp. 335–354. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-28628-8\\_21](https://doi.org/10.1007/978-3-540-28628-8_21)

35. Katz, J., Ostrovsky, R., Smith, A.: Round efficiency of multi-party computation with a dishonest majority. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 578–595. Springer, Heidelberg (2003). [https://doi.org/10.1007/3-540-39200-9\\_36](https://doi.org/10.1007/3-540-39200-9_36)
36. Khurana, D.: Non-interactive distributional indistinguishability (NIDI) and non-malleable commitments. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12698, pp. 186–215. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77883-5\\_7](https://doi.org/10.1007/978-3-030-77883-5_7)
37. Kiyoshima, S., Manabe, Y., Okamoto, T.: Constant-round black-box construction of composable multi-party computation protocol. In: TCC, pp. 343–367 (2014)
38. Lin, H., Pass, R., Soni, P.: Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In: FOCS, pp. 576–587 (2017)
39. Pandey, O., Pass, R., Vaikuntanathan, V.: Adaptive one-way functions and applications. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 57–74. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-85174-5\\_4](https://doi.org/10.1007/978-3-540-85174-5_4)
40. Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 160–176. Springer, Heidelberg (2003). [https://doi.org/10.1007/3-540-39200-9\\_10](https://doi.org/10.1007/3-540-39200-9_10)
41. Sahai, A., Vadhan, S.P.: A complete promise problem for statistical zero-knowledge. In: FOCS, pp. 448–457 (1997)
42. Wee, H.: Black-box, round-efficient secure computation via non-malleability amplification. In: FOCS, pp. 531–540 (2010)





# Minimizing Setup in Broadcast-Optimal Two Round MPC

Ivan Damgård<sup>1</sup>, Divya Ravi<sup>1</sup>, Luisa Siniscalchi<sup>2</sup>(✉), and Sophia Yakoubov<sup>1</sup>

<sup>1</sup> Aarhus University, Aarhus, Denmark

{ivan,divya,sophia.yakoubov}@cs.au.dk

<sup>2</sup> Technical University of Denmark, Kongens Lyngby, Denmark

luisi@dtu.dk

**Abstract.** In this paper we consider two-round secure computation protocols which use different communication channels in different rounds: namely, protocols where broadcast is available in neither round, both rounds, only the first round, or only the second round. The prior works of Cohen, Garay and Zikas (Eurocrypt 2020) and Damgård, Magri, Ravi, Siniscalchi and Yakoubov (Crypto 2021) give tight characterizations of which security guarantees are achievable for various thresholds in each communication structure .

In this work, we introduce a new security notion, namely, *selective identifiable abort*, which guarantees that every honest party either obtains the output, or aborts identifying one corrupt party (where honest parties may potentially identify different corrupted parties). We investigate what broadcast patterns in two-round MPC allow achieving this guarantee across various settings (such as with or without PKI, with or without an honest majority).

Further, we determine what is possible in the honest majority setting *without* a PKI, closing a question left open by Damgård *et al.* We show that without a PKI, having an honest majority does not make it possible to achieve stronger security guarantees compared to the dishonest majority setting. However, if two-thirds of the parties are guaranteed to be honest, *identifiable abort* is additionally achievable using broadcast only in the second round.

We use fundamentally different techniques from the previous works to avoid relying on private communication in the first round when a PKI is not available, since assuming such private channels without the availability of public encryption keys is unrealistic. We also show that, somewhat surprisingly, the availability of private channels in the first round does not enable stronger security guarantees unless the corruption threshold is one.

**Keywords:** Secure computation · Round complexity · Minimal setup

---

S. Yakoubov—Funded in part by the European Research Council (ERC) under the European Unions’s Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO) and No 803096 (SPEC).

© International Association for Cryptologic Research 2023

C. Hazay and M. Stam (Eds.): EUROCRYPT 2023, LNCS 14005, pp. 129–158, 2023.

[https://doi.org/10.1007/978-3-031-30617-4\\_5](https://doi.org/10.1007/978-3-031-30617-4_5)

# 1 Introduction

It is known that secure computation is possible in two rounds (whereas one round is clearly not enough). However, most known two-round protocols either only achieve the weakest security guarantee (selective abort) [1], or achieve the best-possible guarantee in their setting by making use of a broadcast channel in both rounds [4, 12, 15]. Implementing broadcast via a protocol among the parties makes no sense in this setting, as the resulting protocol would require much more than two rounds. However, broadcast can also be done using physical assumptions or external services such as blockchains. This typically means that broadcast is expensive and/or slow, so it is important to try to minimize the usage of broadcast (while achieving as strong a security guarantee as possible).

Before discussing previous work in this direction and our contribution, we establish some useful terminology.

## 1.1 Terminology

In this work, we categorize protocols in terms of (a) the kinds of communication required in each round, (b) the security guarantees they achieve, (c) the setup they require, and (d) the corruption threshold  $t$  they support. We will use shorthand for all of these classifications to make our discussions less cumbersome.

*Communication Structure.* We refer to protocols that use two rounds of broadcast as BC-BC; protocols that use broadcast in the first round only as BC-P2P; protocols that use broadcast in the second round only as P2P-BC; and protocols that don't use broadcast at all as P2P-P2P.

Note that, when no PKI is available, it is not realistic to assume *private* channels in the first round since it is unclear how such private channels would be realized in practice without public keys. Therefore, in what follows, "P2P" in the first round refers to *open* peer-to-peer channels which an adversary can listen in on<sup>1</sup> – unless we explicitly state otherwise. We do assume the availability of private channels in the second round, since one can broadcast (or send over peer-to-peer channels) an encryption under a public key received in the first round.

*Security Guarantees.* There are six notions of security that a secure computation protocol could hope to achieve, described informally below.

**Selective Abort (SA):** A secure computation protocol achieves *selective abort* if every honest party either obtains the output, or aborts.

**Selective Identifiable Abort (SIA):** A secure computation protocol achieves *selective identifiable abort* if every honest party either obtains the output, or aborts identifying one corrupt party (where the corrupt party identified by different honest parties may potentially be different).

---

<sup>1</sup> We do assume that the peer-to-peer channels are authenticated.

**Unanimous Abort (UA):** A secure computation protocol achieves *unanimous abort* if either *all* honest parties obtain the output, or they all (unanimously) abort.

**Identifiable Abort (IA):** A secure computation protocol achieves *identifiable abort* if either all honest parties obtain the output, or they all (unanimously) abort, *identifying one corrupt party*.

**Fairness (FAIR):** A secure computation protocol achieves *fairness* if either all parties obtain the output, or none of them do. In particular, an adversary cannot learn the output if the honest parties do not also learn it.

**Guaranteed Output Delivery (GOD):** A secure computation protocol achieves *guaranteed output delivery* if all honest parties will learn the computation output no matter what the adversary does.

All the notions above, except SIA have been studied previously. This new notion that we introduce here is strictly stronger than selective abort and incomparable with unanimous abort but weaker than identifiable abort (which demands that all honest parties must be in agreement). Selective abort is the weakest notion of security, and is implied by all the others; guaranteed output delivery is the strongest, and implies all the others. Notably, fairness and identifiable abort are incomparable; selective identifiable abort and unanimous abort are incomparable as well.

We believe the notion of SIA is an interesting conceptual contribution. Unlike SA security where an honest party who aborts would only know that someone misbehaved but has no idea who, SIA guarantees that the honest party learns the identity of someone, who behaved incorrectly. This gives the honest party an option of taking some action against this (locally identified) corrupt party, such as refusing to collaborate in other contexts, withholding payment etc. One may also consider a setting where misbehavior due to a technical error is much more likely than a malicious attack. This could be the case, for instance, if parties are considered reliable and with a good level of system security, implying that the weak point is rather the software running the protocol. In such a case, SIA can be used to help find where the error was in case of an abort, whereas SA will give you no such help.

Of course, it would be preferable to achieve IA rather than SIA; but this may not be possible in all settings. The partial identifiability offered by SIA can be quite useful in such settings (where SIA is possible and IA is not), since SIA does not allow anyone to accuse anyone, in particular, honest players will never accuse each other. This means that, in an honest majority setting, if a party has a conflict with more than  $t$  parties, then that party is definitely corrupt. Therefore, if several secure computations are done in a setting where SIA is possible (and IA is not), we can accumulate the conflicts, and may eventually be able to identify a corrupt party such that everyone agrees (namely if that party has a conflict with more than  $t$  parties). This will not completely prevent adversarial behavior, but it may limit the number of honest parties that are forced to abort. Such scenarios highlight why SIA is a useful notion to study.

*Setup.* The following forms of setup, from strongest to weakest, are commonly considered in the MPC literature:

- Correlated randomness (CR)**, where the parties are given input-independent secrets which may be correlated,
- A public key infrastructure (PKI)**, where each party has an independent honestly generated<sup>2</sup> public-secret key pair where the public key is known to everyone, and
- A common reference string (CRS)**, where no per-party information is available, but a single trusted reference string is given.

We focus primarily on protocols that only use a CRS, which is the weakest form of setup (except for the extreme case of no setup at all). To make our prose more readable, when talking about e.g. a secure computation protocol that achieves security with identifiable abort given a CRS and uses broadcast in the second round only, we will refer to it as a P2P-BC, IA, CRS protocol. If we additionally want to specify the corruption threshold  $t$  to be  $x$ , we call it a P2P-BC, IA, CRS,  $t \leq x$  protocol.

## 1.2 Prior Work

Cohen, Garay and Zikas [7] initiated the study of two-round secure computation with broadcast available in one, but not both, rounds. They showed that, in the P2P-BC setting, UA is possible even given a dishonest majority, and that it is the strongest achievable guarantee in this setting. They also showed that, in the BC-P2P setting, SA is the strongest achievable security guarantee given a dishonest majority.

The subsequent work by Damgård, Magri, Ravi, Siniscalchi and Yakoubov [9] continued this line of inquiry, focusing on the honest majority setting. They showed that given an honest majority, in the P2P-BC setting IA is achievable (but fairness is not), and in the BC-P2P setting, the strongest security guarantee—GOD—is achievable.

The constructions of Cohen *et al.* do not explicitly use a PKI, but they do rely on private communication in the first round, which in practice requires a PKI, as discussed above. The constructions of Damgård *et al.* rely on a PKI even more heavily. The natural open question therefore is: what can be done assuming no PKI—only a CRS, and no private communication in the first round?

We note that the recent work of Goel, Jain, Prabhakaran and Raghunath [14] considers instead the plain model or the availability only of a bare PKI (where it is assumed that corrupt parties may generate their public key maliciously). They show that in plain model, in the absence of private channels, no secure computation is possible even given an honest majority. Further, given broadcast (in both rounds) IA is impossible in the plain model, while the strongest guarantee of GOD is feasible in the bare PKI model. Our model is incomparable to that

<sup>2</sup> Throughout this paper, we use the term ‘PKI’ to refer to a ‘trusted PKI’, where the PKI keys are assumed to be honestly generated for all parties.

of Goel *et al.* since we consider the availability of a CRS, and communication patterns where broadcast is limited to one of the two rounds.

To the best of our knowledge, the notion of selective identifiable abort is not discussed in previous work.

### 1.3 Our Contributions

We summarize the contributions of our work in two broad categories, described below.

#### 1.3.1 Introduction of SIA

We introduce and formalize a new security notion of MPC protocols, that we refer to as selective identifiable abort (SIA). Further, we investigate the feasibility of two-round SIA MPC protocols with different broadcast patterns for various settings – with or without PKI, and with or without honest majority. As it turns out, SIA is an interesting notion because it can be achieved in cases where previously only weaker or incomparable notions were known to be possible. Notably, for the P2P-P2P or BC-P2P and  $t < n/3$  settings, SIA can be achieved and is the best possible guarantee, where previously only selective abort was known. Note that, with only selective abort, stopping honest players from getting the output is basically without consequences for the adversary, while with SIA, each honest player will identify at least one corrupt player.

In the following we explain all the results on SIA in more detail: In Theorem 3, we show that any BC-BC (respectively P2P-BC) protocol (with some additional properties) can be transformed to an SIA protocol for the same corruption threshold where the second round communication is over peer-to-peer channels. Plugging in the appropriate IA protocols to this theorem yields several positive results, summarized in Table 1 (for the CRS setting) and Sect. 1.4.3 (for the PKI setting). Namely, we obtain that when we assume only a CRS and no private communication in the first round, SIA is achievable in the P2P-P2P setting with  $t < \frac{n}{3}$  and in the BC-P2P setting with  $t < n$ ; finally when a PKI is available, SIA is also possible in the P2P-P2P setting with  $t < \frac{n}{2}$ .

In light of the above, what remains to be investigated are patterns where broadcast is not available in the first-round, for settings with only a CRS and honest majority ( $\frac{n}{2} > t \geq \frac{n}{3}$ ); and settings with PKI and dishonest majority ( $\frac{n}{2} \leq t < n$ ).

In the CRS only setting, we show that P2P-BC, SIA is impossible to achieve even with an honest majority (Theorem 4, this result holds even when the first-round communication is private). Finally, we observe that the impossibility of P2P-BC, IA, PKI protocols for  $t < n$  in [7] can be extended to SIA as well (elaborated in the full version [10]).

#### 1.3.2 Complete Characterization of Two-Round MPC in the CRS Model with Honest Majority

Assuming only a CRS and no private communication in the first round, we give a complete characterization of what can be done in two rounds with respect to all the other security guarantees and different broadcast patterns.

In a nutshell, we show that assuming only a CRS, an honest majority does not give much of an advantage over a dishonest majority: regardless of the corruption threshold, IA continues to remain impossible in the P2P-BC setting (directly follows from impossibility of SIA in this setting, Theorem 4) and in the BC-P2P setting, UA continues to remain impossible (Theorem 5)<sup>3</sup>. The latter extends the impossibility result of Patra and Ravi [20], which holds for  $n \leq 3t$  (but does not hold for  $t > 1$  and any  $n$ ).

However, if at least two thirds of the parties are honest, in the P2P-BC setting IA is additionally possible (Theorem 2). To show this we give a construction based on a new primitive called *one-or-nothing secret sharing with intermediaries* (adapted from one-or-nothing secret sharing [9]), which may be of independent interest.

Most of our lower bounds hold even given private communication in the first round; however, our constructions do not require it. This shows that surprisingly, in most cases, having private communication in the first round cannot help achieve stronger guarantees.

The one exception is the case where the adversary can only corrupt one party (that is,  $t = 1$ ); for  $t = 1$  and  $n \geq 4$ , guaranteed output delivery can be achieved given private channels in the first round [17, 18] even when broadcast is completely unavailable. However, we show that without private channels in the first round fairness (and thus also guaranteed output delivery) is unachievable, even if broadcast is available in both rounds and the adversary corrupts only one participant<sup>4</sup>. We also show that without private channels in the first round, if broadcast is unavailable in the second round, unanimous abort is unachievable.

Finally, we make a relatively simple observation, showing that the positive results from Cohen *et al.* still hold, even without private communication in the first round.

We summarize our findings in Table 1, and the special case of  $t = 1$  in Table 2.

## 1.4 Technical Overview

In Sect. 1.4.1, we summarize our lower bounds; in Sect. 1.4.2, we summarize our constructions. These results assume a setup with CRS only. Lastly, in Sect. 1.4.3, we summarize the results related to SIA, when PKI is available.

---

<sup>3</sup> Given an additional round of communication instead of a PKI, things look different; Badrinarayanan *et al.* [2] study broadcast-optimal *three-round* MPC with GOD given an honest majority and CRS, and show that GOD is achievable in the BC-BC-P2P setting.

<sup>4</sup> This strengthens the fairness impossibility result of Gordon *et al.* [15] which holds for  $n \leq 3t$ .

**Table 1.** Feasibility and impossibility for two-round MPC with different guarantees and broadcast patterns when only a CRS is available (but no PKI or correlated randomness). The R1 column describes whether broadcast is available in round 1; the R2 column describes whether broadcast is available in round 2. In our constructions, round 1 communications are not private; negative results hold even with private round 1 communications. Arrows indicate implication: the possibility of a stronger security guarantee implies the possibility of weaker ones in the same setting, and the impossibility of a weaker guarantee implies the impossibility of stronger ones in the same setting. Beige table cells are lower bounds; green table cells are upper bounds.

Broadcast Pattern		$t$	Selective Abort (SA)	Selective Identifiable Abort (SIA)	Unanimous Abort (UA)	Identifiable Abort (IA)	Fairness (FAIR)	Guaranteed Output Delivery (GOD)
R1	R2							
BC	BC	$\frac{n}{2} \leq t < n$	✓	✓	✓	✓[7] w.m.c	✗[6]	✗
P2P	BC		✓	✗[7] (see [10] for details)	✓[7] w.m.c	✗	✗	✗
BC	P2P		✓	✓[7] w.m.c, last round P2P (Theorem 3)	✗[7]	✗	✗	✗
P2P	P2P		✓[7] w.m.c	✗	✗	✗	✗	✗
BC	BC	$\frac{n}{3} \leq t < \frac{n}{2}$	✓	✓	✓	✓[7] w.m.c	✗[15, 20]	✗
P2P	BC		✓	✗Theorem 4	✓[7] w.m.c	✗	✗	✗
BC	P2P		✓	✓[7] w.m.c, last round P2P (Theorem 3)	✗[20]	✗	✗[15, 20]	✗
P2P	P2P		✓[7] w.m.c	✗	✗	✗[8]	✗	✗[19]
BC	BC	$t < \frac{n}{3}$	✓	✓	✓	✓[7] w.m.c	✗ for $t > 1$ [13]	✗ for $t > 1$
P2P	BC		✓	✓	✓[7] w.m.c	✓ Theorem 2	✗ for $t > 1$ [13]	✗ for $t > 1$
BC	P2P		✓	✓[7] w.m.c, last round P2P (Theorem 3)	✗ for $t > 1$ (Theorem 5)	✗ for $t > 1$	✗ for $t > 1$ [13]	✗ for $t > 1$
P2P	P2P		✓[7] w.m.c	✓ Theorem 2, last round P2P (Theorem 3)	✗ for $t > 1$ [9]	✗ for $t > 1$	✗ for $t > 1$	✗ for $t > 1$

**Table 2.** Feasibility and impossibility for two-round MPC with different guarantees and broadcast patterns when only a CRS is available, when  $t = 1$ . We refer to Table 1 for the cases already covered therein.

Broadcast Pattern		$t$	Selective Abort (SA)	Selective Identifiable Abort (SIA)	Unanimous Abort (UA)	Identifiable Abort (IA)	Fairness (FAIR)	Guaranteed Output Delivery (GOD)
R1	R2							
<b>The <math>t = 1</math> Case</b>								
<b>Without Private Channels in Round 1:</b>								
BC	BC	$t = 1, n > 1$	Table 1	Table 1	Table 1	Table 1	$\times$ Cor 2 [10]	$\rightarrow \times$
P2P	BC		Table 1	Table 1	Table 1	Table 1	$\times$	$\rightarrow \times$
BC	P2P		Table 1	Table 1	$\times$ Cor 1 [10]	$\rightarrow \times$	$\times$ Cor 1, 2 [10]	$\rightarrow \times$
P2P	P2P		Table 1	Table 1	$\times$	$\rightarrow \times$	$\times$	$\rightarrow \times$
<b>With Private Channels in Round 1:</b>								
Any	$t = 1, n = 4$							
	$t = 1, n \geq 5$							

### 1.4.1 Lower Bounds

We present several lower bounds, some of which hold even when private channels are available in the first round. This is in contrast to our constructions which avoid the use of private channels before the parties had a chance to exchange public keys.

*With Private Channels.* In Sect. 4, we present two main lower bounds that hold even if private channels are available in the first round.

Our first lower bound (Theorem 4) shows that P2P-BC, SIA, CRS protocol is impossible when  $n \leq 3t$ . To show this, we consider a hypothetical 3-party P2P-BC, SIA, CRS protocol where an adversary who controls just one party, say  $P$  behaves inconsistently over the first-round peer-to-peer channels and then chooses to act in the second round based on the information sent to one of the honest parties, say  $P'$ . Then, SIA guarantees that  $P'$  must compute the output even though she finds the pair of remaining parties in conflict, as she cannot decide whom to blame. This makes the protocol vulnerable to an attack by potentially corrupt  $P'$  who can simulate this kind of conflict in her head by recomputing the messages of  $P$  based on inputs of her choice. Infact, this argument can be extended for  $n \leq 3t$ .

Our second lower bound (Theorem 5) shows that BC-P2P, UA, CRS protocol is impossible when  $t > 1$ . To show this, we argue that in any hypothetical BC-P2P, UA, CRS protocol, an adversary who is able to control just two parties is able to perform an even more powerful attack: after execution, she is able to recompute the function output locally on corrupt party inputs of her choice (together with the same fixed set of honest party inputs). This is called a *residual function attack*. This completes the overview of the lower bounds that hold when private channels are present.

When  $t = 1$ , we show that the availability of private channels makes a difference. When private channels are available in the first round, the strongest guarantee—guaranteed output delivery—is known to be achievable as long as



$n \geq 4$  [17, 18]. However, we show in the full version [10] and outline below that without private channels in the first round, the landscape is quite different.

*Without Private Channels.* In this setting, an adversary can observe all messages sent by an honest party  $P$  in the first round; so, those first-round messages cannot suffice to compute the function on  $P$ 's input— $P$ 's second-round messages are crucially necessary for this. If  $P$ 's first-round messages were enough, the adversary would be able to mount a residual function attack: given  $P$ 's first-round messages, the adversary would be able to compute the function on  $P$ 's input (along with inputs of her choice on behalf of the other parties) in her head, by simulating all the other parties. However, if we aim for either unanimous abort (without use of broadcast in the second round) or fairness, we can also argue that  $P$ 's second-round messages *cannot* be necessary. If we would like to achieve unanimous abort without use of broadcast in the second round, it is important that the adversary not be able to break unanimity by sending different second-round messages to different parties. If we would like to achieve fairness, it is similarly important that the adversary not be able to deny the honest parties access to the output by withholding her second-round messages. So, to achieve either of those goals, the second-round message both must and cannot matter; we thus rule out BC-P2P, UA, CRS protocols (Cor 1 in [10]) and BC-BC, FAIR, CRS protocols (Cor 2 in [10]) when no private channels are available in the first round.

### 1.4.2 Upper Bounds

*Feasibility of P2P-BC, IA, CRS when  $t < \frac{n}{3}$*  In Sect. 3.2, we present our main positive result, which is a P2P-BC, IA, CRS,  $t < \frac{n}{3}$  construction (Fig. 2). Our construction builds on the construction of Damgård *et al.* [9] (which, in turn, builds on the construction of Cohen *et al.* [7]). Like those prior works, we take a protocol that requires two rounds of broadcast, and compile it. Since broadcast is only available in the second round, the key is to ensure that a corrupt party can't break the security of the underlying protocol by sending inconsistent messages to different honest parties in the first round. The solution is to delay computation of the second round messages until parties are sure they agree on what was said in the first round.

Following previous work, we do this by having each party  $G$  garble her second-message function (which takes as input all the first-round messages that party expects to receive) and broadcast that garbled circuit in the second round.  $G$  additionally secret shares all of the labels for her garbled circuit. We can get identifiable abort from this if we make sure that one of two things happen: (a) sufficiently many parties receive a given first-round message bit coming from a sender  $S$ , implying that the label corresponding to that bit is reconstructed (unanimously, over broadcast); or (b) someone is unanimously identified as a cheater. (Of course, two labels for the same input wire should never be reconstructed, since this would compromise the security of the garbled circuit scheme.)

To achieve this, Damgård *et al.* introduce (and use in their construction) the notion of *one-or-nothing secret sharing*. Unfortunately, this primitive crucially relies on a PKI: in the second round, each player must be able to prove that she received a certain message from  $S$  in the first round (or abstain if she received nothing). Given a PKI, this can be done by having  $S$  sign her first-round messages. Of course, without a PKI, this cannot work as there is no time to agree on public keys.

Therefore, without a PKI, we need a different approach. The approach we use is instead to check in the second round whether there is sufficient consensus among the parties about what  $S$  sent in the first round, and only reconstruct the corresponding labels if this is the case. To this end, we define a new primitive in the CRS model called *one-or-nothing secret sharing with intermediaries*. In such a scheme, each garbler  $G$  performs two layers of Shamir sharing: first, each label is shared, creating for each party  $R$  a share  $s_R$ . Second, each  $s_R$  is shared among all parties. Everyone now acts as intermediaries, and passes their sub-shares of  $s_R$  on to  $R$  in the second round. This ensures that a corrupt  $G$  cannot fail to deliver a share to  $R$ , since  $G$  cannot fail to communicate with more than  $t$  intermediaries without being identified. Simultaneously, each participant  $R$  broadcasts a message enabling the public recovery of only the label share corresponding to what she received from  $S$  in the first round. Enough shares for a given label are only recoverable if enough participants received the same bit from  $S$ , implicitly implementing the consensus check we mentioned above.

There is one final caveat we need to take care of: the standard network model assumes peer-to-peer “open” channels where the adversary can observe all messages sent. With a PKI, we can make use of private channels (even in the first round), by using public-key encryption (PKE). However, in the absence of a PKI, this makes little sense, so we should *not* use private channels in the first round. Under this constraint we cannot send shares of secrets in the first round. So, we need to figure out a way for  $G$  to send sub-shares of  $R$ 's share  $s_R$  to intermediaries, and for intermediaries to pass these sub-shares on to  $R$ , in a *single* round of broadcast.

The approach we use is as follows: in the second round, for each sub-share of  $s_R$  intended for intermediary  $I$ ,  $G$  will broadcast an encryption  $c$  of that sub-share, under a public key received from  $I$  in the first round. Simultaneously,  $I$  passes on all of sub-shares to  $R$  by broadcasting *transfer keys*. Depending on which value should be decrypted,  $R$  broadcasts the relevant decryption key which enables the recovery of the corresponding plaintext. We informally refer to this approach as *transferrable encryption*, where a party is able to transfer decryption capabilities to another, even without first seeing the ciphertext in question.

Our construction of *one-or-nothing secret sharing with intermediaries* relies on a CPA-secure PKE scheme and non-interactive zero-knowledge (NIZK) proof system. This is used as a building block in our P2P-BC, IA, CRS,  $3t < n$  construction (formalized in Fig. 2) following the above blueprint.

*Modifying Prior P2P-BC Constructions.* The work of Cohen *et al.* gives a construction in the P2P-BC and P2P-P2P settings that uses only a CRS (not a

PKI); however, they use private communication in the first round. We observe that we can modify their construction in a straightforward way to only use *public* peer-to-peer communication in the first round, which is more realistic without a PKI. Their construction is a compiler, and in the first round, two things are sent: messages from the underlying construction; and (full-threshold) secret shares of garbled circuit labels, which need to be communicated privately, and which are then selectively published in the second round. Let's pick an underlying construction that uses public communication only (e.g. the construction of [12]). Now, to avoid private communication in the first round, we modify the protocol to delay secret sharing until the second round. Instead, the only additional thing the parties do in the first round is exchange public encryption keys. Like in our construction (described above), it might look like delaying secret sharing poses a problem, since the share recipients need to broadcast the relevant shares to enable output recovery, but if they only receive their shares in the second (last) round, they don't have time to do this. So, we have the share sender  $G$  encrypt each share meant for receiver  $R$  under a one-time public key belonging to  $R$ . Simultaneously,  $R$  will publish the corresponding secret key if and only if she wishes to enable the reconstruction of that label.<sup>5</sup> In this way, the same guarantees can be achieved without using private communication in the first round.

*Feasibility Results for SIA.* In Sect. 3.3, we argue that a BC-BC protocol (respectively a P2P-BC)  $\Pi_{bc}$  that securely computes  $f$  with identifiable abort can be turned into an SIA protocol  $\Pi$  (with the same corruption threshold) where the second round is run over peer-to-peer channels, as long as  $\Pi_{bc}$  satisfies the following two properties: 1) the simulator can extract inputs from the first-round; 2) it is efficient to check whether a given second-round of the protocol is correct.

The protocol  $\Pi$  works in the same way as  $\Pi_{bc}$ , except that the second round is sent over peer-to-peer channels. Intuitively, the only advantage that the adversary has in  $\Pi$  is to send inconsistent last round messages. However, we argue that this cannot lead to a pair of honest parties obtaining two different non- $\perp$  outputs. This is because of our assumption that the simulator of  $\Pi_{bc}$  extracts input from the first round messages (and say receives the output  $y$  from the ideal functionality). This means that no matter what second round messages the adversary sends in  $\Pi_{bc}$ , the output can never be  $y' \neq \perp$  such that  $y' \neq y$ . More specifically, the adversary's second round messages in  $\Pi_{bc}$  can only determine whether all the honest parties learn  $y$  or all the honest parties learn the identity of a cheater (can be potentially chosen by the adversary during the second broadcast round in  $\Pi_{bc}$ , say, by making a corrupt party stop sending messages or sending invalid messages in the second round). Since these second round messages are now sent over peer-to-peer channels instead (but it is possible to efficiently check their validity), we can conclude that each honest party

<sup>5</sup> Note that the full power of our one-or-nothing secret sharing with intermediaries is not necessary here; in our construction, we only require two levels of sharing and intermediaries in order to achieve *identifiable* abort, while this construction aims only for selective and unanimous abort in the two different settings respectively.

in  $\Pi$  would either learn the output  $y$  or the identity of a cheater (depending on the version of the second round message the adversary sends privately). It may be the case that honest parties learn different cheaters or some of them learn the output  $y$  while others don't; however, this suffices for SIA guarantee.

In Sect. 3.3 we give candidate constructions of BC-BC protocol (respectively a P2P-BC) with identifiable abort, that have the additional properties described above and can thereby be used to yield the BC-P2P (respectively P2P-P2P) SIA upper bounds.

### 1.4.3 Completing the Picture of SIA with PKI

Given that the notion of selective identifiable abort is introduced in this work, we also investigate how it affects the landscape when a PKI is available. This setting was studied by Damgård *et al.* [9] for the case of honest majority and Cohen *et al.* [7] for the case of dishonest majority.

The case of BC-BC is already settled by Cohen *et al.*, who give an IA construction (stronger than SIA) for  $t < n$ , relying just on CRS. Next, we note that our observation in Sect. 3.3 lets us transform the above into an SIA protocol (with the same corruption threshold) where the second round is run over peer-to-peer channels; settling the case of BC-P2P setting.

In the P2P-BC setting, we observe that the impossibility of P2P-BC, IA, PKI protocols for  $t < n$  in Cohen *et al.* can be extended to SIA as well (see full version [10] for details). However, assuming an honest majority ( $t < \frac{n}{2}$ ), feasibility of SIA follows directly from the P2P-BC, IA, PKI construction of Damgård *et al.*. Applying the observation in Sect. 3.3 to this IA construction of Damgård *et al.*, let us achieve SIA for P2P-P2P setting with the same threshold.

This settles the question of feasibility of two-round SIA with various broadcast patterns in the PKI setting.

## 1.5 Broadcast Complexity

In the previous two works, no attempt was made to minimize the broadcast overhead of the compilers. They all require the broadcast of garbled second-message functions, the size of which often scales with the complexity of the function computed, which is potentially large. We observe that a generic broadcast optimization (which is folklore, and has appeared in some previous work [5, 11, 16]) can be applied to any message which is already known to the sender in the first round, but need not be broadcast until the second round. Using this optimization, the size of the additional broadcasts that our compiler—and the compilers of Cohen *et al.* and Damgård *et al.*—becomes independent of the size of the function being computed.

The broadcast optimization is quite straightforward. It enables reliable broadcast of arbitrarily long messages, while only sending fixed-length messages over the broadcast channel in the second round. The dealer sends its message to all the recipients over peer-to-peer channels in the first round. Each recipient then echos the message it received *over peer-to-peer channels* in the second

round. Finally, in the second round, each party also broadcasts a *hash* of the message. If there exists a majority of parties who broadcast the same hash  $h$ , then each honest party outputs a pre-image of  $h$ . (Each party must have received a pre-image of  $h$  because at least one of the broadcasters of  $h$  must be honest.) Otherwise, honest parties blame the dealer. Only hashes are sent over the broadcast channel, and the size of those hashes is independent of the size of the message.

Finally, we note that when applying this optimization to our construction, and that of Cohen *et al.* and Damgård *et al.*, garbled circuits which were previously not broadcast until the second round are now sent (over peer-to-peer channels) in the first round. This necessitates the use of adaptive garbled circuits<sup>6</sup>.

## 2 Secure Multiparty Computation (MPC) Definitions

### 2.1 Security Model

We follow the real/ideal world simulation paradigm and we adopt the security model of Cohen, Garay and Zikas [7]. As in their work, we state our results in a stand-alone setting.<sup>7</sup>

*Real-World.* An  $n$ -party protocol  $\Pi = (P_1, \dots, P_n)$  is an  $n$ -tuple of probabilistic polynomial-time (PPT) interactive Turing machines (ITMs), where each party  $P_i$  is initialized with input  $x_i \in \{0, 1\}^*$  and random coins  $r_i \in \{0, 1\}^*$ . We let  $\mathcal{A}$  denote a special PPT ITM that represents the adversary and that is initialized with input that contains the identities of the corrupt parties, their respective private inputs, and an auxiliary input.

The protocol is executed in rounds (i.e., the protocol is synchronous). Each round consists of the send phase and the receive phase, where parties can respectively send the messages from this round to other parties and receive messages from other parties. In every round parties can communicate either over a broadcast channel or a fully connected peer-to-peer (P2P) network. If peer-to-peer communication occurs in the first round without a PKI, we assume these channels are “open”; that is, the adversary sees all messages sent.<sup>8</sup> In other cases, we assume that these channels can be private, since communications can be encrypted using public keys that are either available via a PKI or exchanged in the first round. In all cases, we assume the channels to be ideally authenticated.

<sup>6</sup> Adaptive garbling schemes [3] remain secure against an adversary who obtains the garbled circuit and then selects the input.

<sup>7</sup> We note that our security proofs can translate to an appropriate (synchronous) composable setting with minimal changes. We also give the formal definition of the new security notion of selective identifiable abort (sl-idabort).

<sup>8</sup> Some of our negative results hold even if private peer-to-peer channels are available in the first round. However, our positive results do not make use of such channels.

During the execution of the protocol, the corrupt parties receive arbitrary instructions from the adversary  $\mathcal{A}$ , while the honest parties faithfully follow the instructions of the protocol. We consider the adversary  $\mathcal{A}$  to be rushing, i.e., during every round the adversary can see the messages the honest parties sent before producing messages from corrupt parties.

At the end of the protocol execution, the honest parties produce output, and the adversary outputs an arbitrary function of the corrupt parties' view. The view of a party during the execution consists of its input, random coins and the messages it sees during the execution.

**Definition 1 (Real-world execution).** *Let  $\Pi = (P_1, \dots, P_n)$  be an  $n$ -party protocol and let  $\mathcal{I} \subseteq [n]$ , of size at most  $t$ , denote the set of indices of the parties corrupted by  $\mathcal{A}$ . The joint execution of  $\Pi$  under  $(\mathcal{A}, \mathcal{I})$  in the real world, on input vector  $x = (x_1, \dots, x_n)$ , auxiliary input  $\mathbf{aux}$  and security parameter  $\lambda$ , denoted  $\text{REAL}_{\Pi, \mathcal{I}, \mathcal{A}(\mathbf{aux})}(x, \lambda)$ , is defined as the output vector of  $P_1, \dots, P_n$  and  $\mathcal{A}(\mathbf{aux})$  resulting from the protocol interaction.*

*Ideal-World.* We describe ideal world executions with selective abort (sl-abort), selective identifiable abort (sl-idabort), unanimous abort (un-abort), identifiable abort (id-abort), fairness (fairness) and guaranteed output delivery (god).

**Definition 2 (Ideal Computation).** *Consider  $\text{type} \in \{\text{sl-abort}, \text{un-abort}, \text{sl-idabort}, \text{id-abort}, \text{fairness}, \text{god}\}$ . Let  $f : (\{0, 1\}^*)^n \rightarrow (\{0, 1\}^*)^n$  be an  $n$ -party function and let  $\mathcal{I} \subseteq [n]$ , of size at most  $t$ , be the set of indices of the corrupt parties. Then, the joint ideal execution of  $f$  under  $(\mathcal{S}, \mathcal{I})$  on input vector  $x = (x_1, \dots, x_n)$ , auxiliary input  $\mathbf{aux}$  to  $\mathcal{S}$  and security parameter  $\lambda$ , denoted  $\text{IDEAL}_{f, \mathcal{I}, \mathcal{S}(\mathbf{aux})}^{\text{type}}(x, \lambda)$ , is defined as the output vector of  $P_1, \dots, P_n$  and  $\mathcal{S}$  resulting from the following ideal process.*

1. Parties send inputs to trusted party: An honest party  $P_i$  sends its input  $x_i$  to the trusted party. The simulator  $\mathcal{S}$  may send to the trusted party arbitrary inputs for the corrupt parties. Let  $x'_i$  be the value actually sent as the input of party  $P_i$ .
2. Trusted party speaks to simulator: The trusted party computes  $(y_1, \dots, y_n) = f(x'_1, \dots, x'_n)$ . If there are no corrupt parties or  $\text{type} = \text{god}$ , proceed to step 4..
  - (a) If  $\text{type} \in \{\text{sl-abort}, \text{un-abort}, \text{sl-idabort}, \text{id-abort}\}$ : The trusted party sends  $\{y_i\}_{i \in \mathcal{I}}$  to  $\mathcal{S}$ .
  - (b) If  $\text{type} = \text{fairness}$ : The trusted party sends ready to  $\mathcal{S}$ .
3. Simulator  $\mathcal{S}$  responds to trusted party:
  - (a) If  $\text{type} = \text{sl-abort}$ : The simulator  $\mathcal{S}$  can select a set of parties that will not get the output as  $\mathcal{J} \subseteq [n] \setminus \mathcal{I}$ . (Note that  $\mathcal{J}$  can be empty, allowing all parties to obtain the output.) It sends  $(\text{abort}, \mathcal{J})$  to the trusted party.
  - (b) If  $\text{type} \in \{\text{un-abort}, \text{fairness}\}$ : The simulator can send abort to the trusted party. If it does, we take  $\mathcal{J} = [n] \setminus \mathcal{I}$ .
  - (c) If  $\text{type} = \text{sl-idabort}$ : The simulator  $\mathcal{S}$  can select a set of parties that will not get the output as  $\mathcal{J} \subseteq [n] \setminus \mathcal{I}$ . (Note that  $\mathcal{J}$  can be empty, allowing all parties to obtain the output.) For each party  $j$  in  $\mathcal{J}$ , the adversary

selects a corrupt party  $i_j^* \in \mathcal{I}$  who will be blamed by party  $j$ . It sends  $(\text{abort}, \mathcal{J}, \{j, i_j^*\}_{j \in \mathcal{J}})$  to the trusted party.

- (d) If  $\text{type} = \text{id-abort}$ : If it chooses to abort, the simulator  $\mathcal{S}$  can select a corrupt party  $i^* \in \mathcal{I}$  who will be blamed, and send  $(\text{abort}, i^*)$  to the trusted party. If it does, we take  $\mathcal{J} = [n] \setminus \mathcal{I}$ .

4. Trusted party answers parties:

- (a) If the trusted party got **abort** from the simulator  $\mathcal{S}$ ,

i. It sets the abort message  $\text{abortmsg}$ , as follows:

- if  $\text{type} \in \{\text{sl-abort}, \text{un-abort}, \text{fairness}\}$ , we let  $\text{abortmsg} = \perp$ .
- if  $\text{type} = \text{sl-idabort}$ , we let  $\text{abortmsg} = \{\text{abortmsg}_j\}_{j \in \mathcal{J}} = (\perp, i_j^*)_{j \in \mathcal{J}}$ .
- if  $\text{type} = \text{id-abort}$ , we let  $\text{abortmsg} = (\perp, i^*)$ .

ii. The trusted party sends  $y_j$  to every party  $P_j$ ,  $j \in [n] \setminus \mathcal{J}$ .

If  $\text{type} = \text{sl-idabort}$ , the trusted party then sends  $\text{abortmsg}_j$  to each party  $P_j$ ,  $j \in \mathcal{J}$ ; otherwise, the trusted party sends  $\text{abortmsg}$  to every party  $P_j$ ,  $j \in \mathcal{J}$ .

Note that, if  $\text{type} = \text{god}$ , we will never be in this setting, since  $\mathcal{S}$  was not allowed to ask for an abort.

- (b) Otherwise, it sends  $y$  to every  $P_j$ ,  $j \in [n]$ .

5. Outputs: Honest parties always output the message received from the trusted party while the corrupt parties output nothing. The simulator  $\mathcal{S}$  outputs an arbitrary function of the initial inputs  $\{x_i\}_{i \in \mathcal{I}}$ , the messages received by the corrupt parties from the trusted party and its auxiliary input.

*Security Definitions.* We now define the security notion for protocols.

**Definition 3** Consider  $\text{type} \in \{\text{sl-abort}, \text{un-abort}, \text{sl-idabort}, \text{id-abort}, \text{fairness}, \text{god}\}$ . Let  $f : (\{0, 1\}^*)^n \rightarrow (\{0, 1\}^*)^n$  be an  $n$ -party function. A protocol  $\Pi$   $t$ -securely computes the function  $f$  with  $\text{type}$  security if for every PPT real-world adversary  $\mathcal{A}$  with auxiliary input  $\text{aux}$ , there exists a PPT simulator  $\mathcal{S}$  such that for every  $\mathcal{I} \subseteq [n]$  of size at most  $t$ , for all  $x \in (\{0, 1\}^*)^n$ , for all large enough  $\lambda \in \mathbb{N}$ , it holds that

$$\text{REAL}_{\Pi, \mathcal{I}, \mathcal{A}(\text{aux})}(x, \lambda) \stackrel{c}{=} \text{IDEAL}_{f, \mathcal{I}, \mathcal{S}(\text{aux})}^{\text{type}}(x, \lambda).$$

## 2.2 Notation

In this paper, we focus on two-round secure computation protocols. Rather than viewing a protocol  $\Pi$  as an  $n$ -tuple of interactive Turing machines, it is convenient to view each Turing machine as a sequence of three algorithms:  $\text{frst-msg}_i$ , to compute  $P_i$ 's first messages to its peers;  $\text{snd-msg}_i$ , to compute  $P_i$ 's second messages; and  $\text{output}_i$ , to compute  $P_i$ 's output. Thus, a protocol  $\Pi$  can be defined as  $\{(\text{frst-msg}_i, \text{snd-msg}_i, \text{output}_i)\}_{i \in [n]}$ .

The syntax of the algorithms is as follows:

- $\text{frst-msg}_i(x_i, r_i) \rightarrow (\text{msg}_{i \rightarrow 1}^1, \dots, \text{msg}_{i \rightarrow n}^1)$  produces the first-round messages of party  $P_i$  to all parties. Note that a party's message to itself can be considered to be its state.

- $\text{snd-msg}_i(x_i, r_i, \text{msg}_{1 \rightarrow i}^1, \dots, \text{msg}_{n \rightarrow i}^1) \rightarrow (\text{msg}_{i \rightarrow 1}^2, \dots, \text{msg}_{i \rightarrow n}^2)$  produces the second-round messages of party  $P_i$  to all parties.
- $\text{output}_i(x_i, r_i, \text{msg}_{1 \rightarrow i}^1, \dots, \text{msg}_{n \rightarrow i}^1, \text{msg}_{1 \rightarrow i}^2, \dots, \text{msg}_{n \rightarrow i}^2) \rightarrow y_i$  produces the output returned to party  $P_i$ .

We implicitly assume that all of these algorithms also take a CRS as input when one is available.

When the first round is over broadcast channels, we consider  $\text{frst-msg}_i$  to return only one message— $\text{msg}_i^1$ . Similarly, when the second round is over broadcast channels, we consider  $\text{snd-msg}_i$  to return only  $\text{msg}_i^2$ .

Throughout our negative results, we omit the randomness  $r$ , and instead focus on deterministic protocols, modeling the randomness implicitly as part of the algorithm.

### 3 Upper Bounds

We begin with a description of our new primitive, one-or-nothing secret sharing with intermediaries, which is used as a building block in our IA construction. Next, we present our positive results for IA and SIA.

#### 3.1 One-or-Nothing Secret Sharing with Intermediaries

Damgård *et al.* [9] introduce one-or-nothing secret sharing, which allows a dealer to share a vector of secrets in such a way that during reconstruction, at most one of the secrets is recovered (the share holders essentially vote on which one). The correctness guarantee is that if sufficiently many share holders vote for a certain index, and no-one votes against that index (though some parties may equivocate), the value at that index is recovered; the security guarantee is that if at least one party votes for a certain index, the adversary learns nothing about the values at any *other* index. Damgård *et al.* present two versions of this primitive: the default version, and a *non-interactive* version, where parties can vote even if they have not received a share from the dealer. This is done by assuming the dealer shares secret keys with each party, which can be realized via non-interactive key exchange, using a PKI.

Unfortunately, this non-interactive one-or-nothing secret sharing tool (referred to as `1or0`) does not extend to a setting where no PKI is available. In the absence of PKI, the main challenge is to ensure that the share intended for a party, say  $P$ , gets delivered (so that her share corresponding to the secret at the index she votes for can be recovered). We achieve this by modeling the fact that other parties can be intermediaries who aid this share transfer. For the setting where only a CRS is available, we propose a new variant of one-or-nothing secret sharing: namely, one-or-nothing secret sharing with intermediaries (referred to as `1or0wi`).

In order to simplify the presentation of our P2P-BC, IA, CRS construction, we define one-or-nothing secret sharing with intermediaries as a *maliciously-secure* primitive. The first round of our protocol is reserved for the exchange



of public keys, so sharing and reconstruction must take place in a single round. The definitions of one-or-nothing secret sharing with intermediaries capture the fact that keys may not have been exchanged consistently, but demand that reconstruction succeeds if blame cannot be assigned. We discuss the syntax, definitions and construction of one-or-nothing secret sharing with intermediaries below.

### 3.1.1 Syntax

A one-or-nothing secret sharing scheme [9] consists of four algorithms: **setup**, **share**, **vote**, and **reconstruct**. **setup** returns a shared secret key belonging to the dealer and one of the receivers; these keys are then used within **share**, and again in **vote**. To make our one-or-nothing secret sharing with intermediaries secure against malicious adversaries, we move to a public-key syntax, which makes it easier to check parties' behavior using zero knowledge proofs. We change **setup** to return a common reference string  $crs$ ; keys are then produced by **keygen**, which creates a key pair for one of the receivers. **share**, **vote** and **reconstruct** now all expect the receivers' public keys as input. The syntax of **reconstruct** is modified to support cheater identification; if sufficiently many (at least  $n-t$ ) parties vote for the same value, then either the secret corresponding to this value will be reconstructed, or a cheating party will be identified. We present the syntax of the maliciously-secure one-or-nothing secret sharing with intermediaries below.

$\mathbf{setup}(1^\lambda) \rightarrow crs$  is an algorithm which takes as input the security parameter and generates the common reference string.

$\mathbf{keygen}(crs) \rightarrow (\mathbf{sk}, \mathbf{pk})$  is an algorithm which takes as input the common reference string and generates a key pair.

$\mathbf{share}(crs, \mathbf{pk}_1, \dots, \mathbf{pk}_n, \mathbf{z}^{(1)}, \dots, \mathbf{z}^{(l)}) \rightarrow s$  is an algorithm run by the dealer  $D$  which takes as input all the parties' public keys, and the  $l$  values that are being shared. It outputs a single share  $s$ .

$\mathbf{vote}(crs, \mathbf{sk}_i, \mathbf{pk}_1, \dots, \mathbf{pk}_n, v_i) \rightarrow \bar{s}_i$  is an algorithm run by party  $i$  which takes as input party  $i$ 's secret key, all the parties' public keys, and a vote  $v_i$ , where  $v_i \in \{1, \dots, l, \perp\}$  can either be an index of a value, or it can be  $\perp$  if party  $i$  is unsure which value it wants to vote for. It returns a ballot  $\bar{s}_i$ .

Note that, to allow **share** and **vote** to be executed in a single round, **vote** does not take as input the share  $s$ .

$\mathbf{reconstruct}(crs, s, (\mathbf{pk}_1, v_1, \bar{s}_1), \dots, (\mathbf{pk}_n, v_n, \bar{s}_n)) \rightarrow \{\mathbf{z}^{(v)}, \perp, \perp_i\}$  is an algorithm which takes as input the output of **share** run by the dealer  $D$ , the outputs of **vote** run by each of the  $n$  parties, as well as their votes, and outputs the value  $\mathbf{z}^{(v)}$  which received a majority of votes, or  $\perp$ , or  $\perp_i$  where  $i$  denotes the identity of a cheater.

### 3.1.2 Security

We require one-or-nothing secret sharing with intermediaries to satisfy *privacy* and *identifiability*, described below. Notice that identifiability naturally

implies correctness. Our definitions of privacy and identifiability both assume that corrupt parties might provide honest parties, including the dealer, with inconsistent or incorrect public keys. Below, we denote the set of  $n$  parties as  $\{D, P_1, \dots, P_{n-1}\}$ , where  $D$  denotes the dealer.

**Informal Definition 1 (1or0wi: Privacy)** *Informally, this property requires that when fewer than  $n - 2t$  honest parties produce their ballot using  $v$ , then the adversary learns nothing about  $\mathbf{z}^{(v)}$ .*

The one-or-nothing secret sharing of Damgård *et al.* [9] additionally required *contradiction-privacy*. This guaranteed the privacy of all secrets when a pair of honest parties produce ballots for different indices. Notably, our one-or-nothing secret sharing with intermediaries does not have this property; however, when  $n > 3t$ , the privacy property implies that at most one secret is reconstructed.<sup>9</sup>

**Informal Definition 2 (1or0wi: Identifiability)** *Informally, this property requires that when at least  $n - t$  parties produce their ballot using the same  $v$ , either `reconstruct` returns  $\mathbf{z}^{(v)}$  or a corrupt party is identified.*

It is easy to see that the identifiability property defined above implies *correctness* (i.e. when all algorithms are executed honestly, if at least  $n - t$  parties produce their ballot using the same  $v$ , `reconstruct` returns  $\mathbf{z}^{(v)}$ ).

We refer to the full version [10] for the formal definitions of privacy and identifiability.

### 3.1.3 Construction

Both the one-or-nothing secret sharing scheme of Damgård *et al.* [9] and our construction of one-or-nothing secret sharing with intermediaries make use of two layers of Shamir secret sharing. However, Damgård *et al.* crucially differ in the way in which the sub-shares for reconstructing a given value are transferred by the shareholders. Because without a PKI a dealer might not communicate reliably/verifiably to all share recipients (as either she or they might be corrupt), in order to achieve identifiability in such scenarios, we introduce a new tool which we informally call *transferrable encryption*.

*Transferrable encryption* allows a sender to encrypt a message to an intermediary, who, even before seeing the ciphertext, can *transfer* the ability to decrypt to another receiver. This can be achieved, for instance, simply by having the intermediary encrypt her (single-use) secret decryption key to the receiver.

We now informally describe the one-or-nothing secret sharing with intermediaries algorithms `keygen`, `share`, `vote`, and `reconstruct`:

<sup>9</sup> If we consider the more general case of  $t' \leq t$  corruptions, the adversary would learn the secret at an index  $v$  only if at least  $(n - t - t')$  honest parties vote for  $v$  (as these along with the  $t'$  ballots known on behalf of the corrupt parties would allow the secret to be reconstructed). Therefore, for the adversary to learn secrets at two different indices, there must exist two disjoint sets of at least  $(n - t - t')$ . This could happen only if  $2(n - t - t') \leq n - t'$ , which implies  $n \leq 2t + t' \leq 3t$  (as  $t' \leq t$ ); which contradicts our assumption of  $n > 3t$ .

1. Informally, **keygen** generates many single-use public-key encryption key pairs for each party  $i$ , designated for transference of decryption power to different parties  $j$ . Each party  $i$  will end up with a key pair  $(\mathbf{sk}_{j \rightarrow i}^{(v)}, \mathbf{pk}_{j \rightarrow i}^{(v)})$  for every party  $j$  and shared value index  $v$ .
2. In the **share** algorithm the dealer threshold secret shares each secret  $\mathbf{z}^{(v)}$  as  $s_1^{(v)}, \dots, s_n^{(v)}$ , and then threshold secret shares each  $s_i^{(v)}$  as  $s_{i \rightarrow 1}^{(v)}, \dots, s_{i \rightarrow n}^{(v)}$ . Then, the dealer broadcasts an encryption of each sub-share  $s_{i \rightarrow j}^{(v)}$  under a key  $\mathbf{pk}_{i \rightarrow j}^{(v)}$  belonging to party  $j$ ; later, during **vote**, party  $j$  will act as an intermediary, and forward that share to party  $i$ .
3. **vote** is divided into two sub-steps (the first of which is independent of the party's vote):
  - (a) Each party  $j$  broadcasts *transfer keys* for each index  $v$  and each other party  $i$  that can be applied to the encryption of  $s_{i \rightarrow j}^{(v)}$  (under party  $j$ 's public key  $\mathbf{pk}_{i \rightarrow j}^{(v)}$ ) to make it decryptable using party  $i$ 's secret decryption key  $\mathbf{sk}_{i \rightarrow i}^{(v)}$ . (Such a transfer key can simply be an encryption of  $\mathbf{sk}_{i \rightarrow j}^{(v)}$  under party  $i$ 's public key  $\mathbf{pk}_{i \rightarrow i}^{(v)}$ .)
  - (b) To vote for the reconstruction of  $\mathbf{z}^{(v)}$ , each party  $i$  broadcasts her relevant secret decryption key  $\mathbf{sk}_{i \rightarrow i}^{(v)}$ .
4. Finally, the **reconstruct** algorithm decrypts all the shares made available through the broadcast of the relevant decryption keys, and reconstructs  $\mathbf{z}^{(v)}$  if at least  $n - t$  votes supported  $v$ ; otherwise, a cheating party is identified.

Finally, to achieve security against an active adversary, each party provides a non-interactive zero-knowledge proof (NIZK) to ensure that each step is honestly computed. Therefore, the **setup** algorithm is also tasked with providing the CRSs required for the NIZKs.

More formally, let  $\text{PKE} = (\mathbf{keygen}, \mathbf{enc}, \mathbf{dec})$  be a public key encryption scheme with CPA security, and let  $\text{NIZK} = (\mathbf{setupZK}, \mathbf{prove}, \mathbf{verify}, \mathbf{simP}, \mathbf{extract})$  be a non-interactive zero-knowledge proof system for the following relations:

$$\mathcal{R}_{\mathbf{keygen}} = \left\{ \begin{array}{l} \phi = \mathbf{pk} \\ w = (\mathbf{sk}, r) \end{array} \middle| (\mathbf{sk}, \mathbf{pk}) \leftarrow \text{PKE.keygen}(1^\lambda; r) \right\},$$

$$\mathcal{R}_{\mathbf{share}} = \left\{ \begin{array}{l} \phi = \{ \mathbf{pk}_{i \rightarrow j}^{(v)}, c_{i \rightarrow j}^{(v)} \}_{v \in [l], i, j \in [n]} \\ w = \left( \begin{array}{l} \{ \mathbf{z}^{(v)}, r^{(v)}, \{ r_i^{(v)}, \\ \{ r_{i \rightarrow j}^{(v)} \}_{j \in [n]} \}_{i \in [n]} \}_{v \in [l]} \end{array} \right) \end{array} \middle| \begin{array}{l} \{ (s_1^{(v)}, \dots, s_n^{(v)}) \leftarrow \text{Shamir.share}(\mathbf{z}^{(v)}; r^{(v)}) \}_{v \in [l]} \\ \wedge \{ (s_{i \rightarrow 1}^{(v)}, \dots, s_{i \rightarrow n}^{(v)}) \leftarrow \text{Shamir.share}(s_i^{(v)}; r_i^{(v)}) \}_{v \in [l], i \in [n]} \\ \wedge \{ c_{i \rightarrow j}^{(v)} \leftarrow \text{PKE.enc}(\mathbf{pk}_{i \rightarrow j}^{(v)}, s_{i \rightarrow j}^{(v)}; r_{i \rightarrow j}^{(v)}) \}_{v \in [l], i, j \in [n]} \end{array} \right\},$$

$$\mathcal{R}_{\text{vote}} = \left\{ \begin{array}{l} \phi = \left( \begin{array}{l} \{\text{pk}_{j \rightarrow j}^{(v)}, \text{pk}_{j \rightarrow i}^{(v)}, \text{tk}_{j \rightarrow i}^{(v)}\}_{v \in [l], j \in [n]}, \\ v_i, \text{sk}_i^{(v_i)} \end{array} \right) \\ w = \left( \begin{array}{l} \{\text{sk}_{j \rightarrow i}^{(v)}, \bar{r}_{j \rightarrow i}^{(v)}, r_j^{(v)}\}_{v \in [l], j \in [n]} \end{array} \right) \end{array} \right\} \left| \begin{array}{l} \{(\text{sk}_{j \rightarrow i}^{(v)}, \text{pk}_{j \rightarrow i}^{(v)}) \leftarrow \text{PKE.keygen}(1^\lambda; \bar{r}_{j \rightarrow i}^{(v)})\}_{j \in [n], v \in [l]} \\ \wedge \{\text{tk}_{j \rightarrow i}^{(v)} \leftarrow \text{PKE.enc}(\text{pk}_{j \rightarrow j}^{(v)}, \text{sk}_{j \rightarrow i}^{(v)}; r_j^{(v)})\}_{v \in [l], j \in [n]} \end{array} \right.$$

Figure 1 describes our one-or-nothing secret sharing with intermediaries (1or0wi) scheme.

Figure 1: Construction of 1or0wi

**setup**( $1^\lambda$ ): Set up and output the common reference strings

$crs_{\text{keygen}} \leftarrow \text{setupZK}(1^\lambda, \mathcal{R}_{\text{keygen}})$ ,  
 $crs_{\text{share}} \leftarrow \text{setupZK}(1^\lambda, \mathcal{R}_{\text{share}})$ , and  
 $crs_{\text{vote}} \leftarrow \text{setupZK}(1^\lambda, \mathcal{R}_{\text{vote}})$

for the zero knowledge proof system. Return  $crs = (crs_{\text{keygen}}, crs_{\text{share}}, crs_{\text{vote}})$ .

**keygen**( $crs$ ), run by party  $i$ :

1. For each  $j \in [n]$  and  $v \in [l]$ ,  $(\text{sk}_{j \rightarrow i}^{(v)}, \text{pk}_{j \rightarrow i}^{(v)}) \leftarrow \text{PKE.keygen}(1^\lambda; \bar{r}_{j \rightarrow i}^{(v)})$ .
2. For each  $j \in [n]$  and  $v \in [l]$ ,  $\pi_{j \rightarrow i}^{(v)} \leftarrow \text{NIZK.prove}(crs_{\text{keygen}}, \phi = \text{pk}_{j \rightarrow i}^{(v)}, w = (\text{sk}_{j \rightarrow i}^{(v)}, \bar{r}_{j \rightarrow i}^{(v)}))$ .
3. Let  $\text{sk}_i = (\{\text{sk}_{j \rightarrow i}^{(v)}, \bar{r}_{j \rightarrow i}^{(v)}\}_{j \in [n], v \in [l]})$ , and  $\text{pk}_i = (\{\text{pk}_{j \rightarrow i}^{(v)}, \pi_{j \rightarrow i}^{(v)}\}_{j \in [n], v \in [l]})$ .
4. Output  $(\text{sk}_i, \text{pk}_i)$ .

**share**( $crs, \text{pk}_1, \dots, \text{pk}_n, \mathbf{z}^{(1)}, \dots, \mathbf{z}^{(l)}$ ), run by the dealer  $D$  (where  $\text{pk}_i = \{\text{pk}_{j \rightarrow i}^{(v)}, \pi_{j \rightarrow i}^{(v)}\}_{j \in [n], v \in [l]}$ ):

1. For each  $v \in [l]$ , compute  $(s_1^{(v)}, \dots, s_n^{(v)}) \leftarrow \text{Shamir.share}(\mathbf{z}^{(v)}; r^{(v)})$  as the threshold sharing of  $\mathbf{z}^{(v)}$  with threshold  $(n - t - 1)$ .
2. For each  $i \in [n]$  and  $v \in [l]$ , compute  $(s_{i \rightarrow 1}^{(v)}, \dots, s_{i \rightarrow n}^{(v)}) \leftarrow \text{Shamir.share}(s_i^{(v)}; r_i^{(v)})$  as the threshold sharing of  $s_i^{(v)}$  with threshold  $(n - 2t - 1)$ .
3. For each  $i, j \in [n]$  and  $v \in [l]$ , compute  $c_{i \rightarrow j}^{(v)} \leftarrow \text{PKE.enc}(\text{pk}_{i \rightarrow j}^{(v)}, s_{i \rightarrow j}^{(v)}; r_{i \rightarrow j}^{(v)})$ .
4. Set
  - $\phi_{\text{share}} = (\{\text{pk}_{i \rightarrow j}^{(v)}, c_{i \rightarrow j}^{(v)}\}_{v \in [l], i, j \in [n]})$  and
  - $w_{\text{share}} = (\{\mathbf{z}^{(v)}, r^{(v)}, \{r_i^{(v)}, \{r_{i \rightarrow j}^{(v)}\}_{j \in [n]}\}_{i \in [n]}\}_{v \in [l]})$ .
 Compute  $\pi_{\text{share}} \leftarrow \text{prove}(crs_{\text{share}}, \phi_{\text{share}}, w_{\text{share}})$ .

5. Set  $s = (\phi_{\text{share}}, \pi_{\text{share}})$  and output  $s$ .

**vote**( $crs, sk_i, pk_1, \dots, pk_n, v_i$ ), run by party  $i$  (where  $pk_i = \{pk_{j \rightarrow i}^{(v)}, \pi_{j \rightarrow i}^{(v)}\}_{j \in [n], v \in [l]}$  and  $sk_i = \{sk_{j \rightarrow i}^{(v)}, \bar{r}_{j \rightarrow i}^{(v)}\}_{j \in [n], v \in [l]}$ ):

1. For each  $v \in [l]$  and  $j \in [n]$ , let  $tk_{j \rightarrow i}^{(v)} \leftarrow \text{PKE.enc}(pk_{j \rightarrow j}^{(v)}, sk_{j \rightarrow i}^{(v)}; r_j^{(v)})$ .
2. Set
  - $\phi_{\text{vote}, i} = (\{pk_{j \rightarrow j}^{(v)}, pk_{j \rightarrow i}^{(v)}, tk_{j \rightarrow i}^{(v)}\}_{v \in [l], j \in [n]}, v_i, sk_{i \rightarrow i}^{(v)})$  <sup>a</sup>
  - $w_{\text{vote}, i} = (\{sk_{j \rightarrow i}^{(v)}, \bar{r}_{j \rightarrow i}^{(v)}, r_j^{(v)}\}_{v \in [l], j \in [n]})$ .
 Compute  $\pi_{\text{vote}, i} \leftarrow \text{prove}(crs_{\text{vote}}, \phi_{\text{vote}, i}, w_{\text{vote}, i})$ .
3. Set  $\bar{s}_i = (\phi_{\text{vote}, i}, \pi_{\text{vote}, i})$  and output  $\bar{s}_i$ .

**reconstruct**( $crs, s, (pk_1, v_1, \bar{s}_1), \dots, (pk_n, v_n, \bar{s}_n)$ ) (where  $s = (\{pk_{i \rightarrow j}^{(v)}, c_{i \rightarrow j}^{(v)}\}_{v \in [l], i, j \in [n]}, \pi_{\text{share}})$ ,  $pk_i = \{pk_{j \rightarrow i}^{(v)}, \pi_{j \rightarrow i}^{(v)}\}_{j \in [n], v \in [l]}$  and  $\bar{s}_i = (\phi_{\text{vote}, i} = (\{pk_{j \rightarrow j}^{(v)}, pk_{j \rightarrow i}^{(v)}, tk_{j \rightarrow i}^{(v)}\}_{v \in [l], j \in [n]}, v_i, sk_{i \rightarrow i}^{(v)}), \pi_{\text{vote}, i})$ ):

Identify the winning vote:

1. If there does not exist a  $v \in \{1, \dots, l\}$  such that at least  $(n - t)$  parties vote for  $v$ , output  $\perp$ . Let  $S_{\text{vote}} \subseteq [n]$  be the set of parties  $i$  such that  $v_i = v$ .

Verify the zero knowledge proofs:

2. For  $i, j \in [n]$ , if  $\text{NIZK.verify}(crs_{\text{keygen}}, \phi = pk_{j \rightarrow i}^{(v)}, \pi_{j \rightarrow i}^{(v)}) = \text{reject}$  (where  $pk_{j \rightarrow i}^{(v)}, \pi_{j \rightarrow i}^{(v)}$  are taken from  $pk_i$ ), return  $\perp_i$ .
3. If  $\text{NIZK.verify}(crs_{\text{share}}, \phi_{\text{share}}, \pi_{\text{share}}) = \text{reject}$  (where  $\phi_{\text{share}}, \pi_{\text{share}}$  are taken from  $s$ ), return  $\perp_D$ .
4. For  $i \in [n]$ , if  $\text{NIZK.verify}(crs_{\text{vote}}, \phi_{\text{vote}, i}, \pi_{\text{vote}, i}) = \text{reject}$  (where  $\phi_{\text{vote}, i}, \pi_{\text{vote}, i}$  are taken from  $\bar{s}_i$ ), return  $\perp_i$ .

Check the consistency of the share, ballots and keys:

5. For  $i \in [n]$ , let  $S'_i \subseteq [n]$  be the set of parties  $j \in [n]$  such that (a)  $pk_{i \rightarrow i}^{(v)}$  is the same in  $pk_i$  and  $\bar{s}_j$ , and (b)  $pk_{j \rightarrow j}^{(v)}$  is the same in  $pk_j$  and  $\bar{s}_i$ . If  $|S'_i| < n - t$ , return  $\perp_i$ .
6. Let  $S_D \subseteq [n]$  be the set of parties  $i$  such that  $\{pk_{j \rightarrow i}^{(v)}\}_{j \in [n]}$  is the same in  $pk_i$  and  $s$ . If  $|S_D| < n - t$ , return  $\perp_D$ .
7. For  $i \in S_{\text{vote}}$ , let  $S_i = S'_i \cap S_D$ . Note that  $|S_i| \geq n - 2t$ .  
For  $j \in S_i$ , we have a ciphertext  $tk_{i \rightarrow j}^{(v)}$  (contained in  $\bar{s}_j$ ), a secret key  $sk_{i \rightarrow i}^{(v)}$  (contained in  $\bar{s}_i$ ) and a ciphertext  $c_{i \rightarrow j}^{(v)}$  (contained in  $s$ ). Let  $sk_{i \rightarrow j}^{(v)} \leftarrow \text{PKE.dec}(sk_{i \rightarrow i}^{(v)}, tk_{i \rightarrow j}^{(v)})$ . Let  $s_{i \rightarrow j}^{(v)} \leftarrow \text{PKE.dec}(sk_{i \rightarrow j}^{(v)}, c_{i \rightarrow j}^{(v)})$ .
8. For each  $i \in S_{\text{vote}}$ , let  $s_i^{(v)} \leftarrow \text{Shamir.reconstruct}(\{s_{i \rightarrow j}^{(v)}\}_{j \in S_i})$ .
9. Output  $z^{(v)} \leftarrow \text{Shamir.reconstruct}(\{s_i^{(v)}\}_{i \in S_{\text{vote}}})$ .

---

*Maliciously secure one-or-nothing secret sharing with intermediaries when  $n > 3t$ .*

---

<sup>a</sup> any string  $m^{(\perp)}$  is to be interpreted as  $\perp$ .

**Theorem 1.** *The construction in Fig. 1 is a maliciously secure one-or-nothing secret sharing with intermediaries when  $n > 3t$  if PKE is a public key encryption scheme with CPA security, and NIZK is a secure non-interactive zero-knowledge proof system.*

The proof appears in the full version [10]

### 3.2 IA Feasibility Result: P2P-BC, IA, $3t < n$

Our upper bounds are based on those of Cohen *et al.* [7] and Damgård *et al.* [9]. They take a BC-BC protocol  $\Pi_{bc}$ , and *compile* it to the P2P-BC setting. The primary challenge here is making sure that corrupt parties cannot break security by sending different messages to honest parties in the first round. Our compiler makes sure that if corrupt party first-round messages are *consistent enough*, honest party second-round messages are produced on the same set of first-round messages; otherwise, a corrupt party is unanimously identified. To achieve this, we (and the prior works) have each party garble her second-message function, which has her own input hardcoded, and takes as input all the first-round messages she receives. Each party also secret-shares all of the labels for her own garbled circuit. In the second round, over broadcast, parties echo the first-round messages they received, distribute their garbled circuit, and contribute to label reconstruction (for everyone’s garbled circuits) corresponding to the first-round messages they received. If there aren’t  $n - t$  parties who all echo the same first-round message from a given  $P_i$ , honest parties abort blaming  $P_i$ ; if there aren’t  $n - t$  parties who all contribute valid ballots for  $P_j$ ’s labels, honest parties abort blaming  $P_j$ . Note that if an (identifiable) abort happens, reconstruction is allowed to fail.

Using Shamir secret sharing with threshold  $s = \frac{3n}{5}$ , this leads to a P2P-BC, IA, CRS protocol with  $t < \frac{n}{5}$ . The reason we have corruption threshold  $t = \frac{n}{5}$  and sharing threshold  $s = \frac{3n}{5}$  is that we have two constraints:

1. In order to prevent the adversary from learning two labels for the same wire by sending different first-round messages to two subsets of the honest parties, we need  $s \geq t + \frac{n-t}{2}$ .
2. In order to ensure that even after (a)  $t$  parties echo a different message from party  $m$  and (b) a different  $t$  parties give bad label shares we still have enough shares to reconstruct, we need  $s < n - 2t$ . (If only  $t$  parties have inconsistent claims with the message sender and a different  $t$  parties have inconsistent claims with the label share dealer, we have no idea who to blame, so we *have to reconstruct!*)

We get

$$\begin{aligned} t + \frac{n-t}{2} &\leq s < n - 2t \\ \Rightarrow t + n &< 2n - 4t \\ \Rightarrow 5t &< n. \end{aligned}$$

However,  $5t < n$  does not match the lower bound from Theorem 4.

To match the lower bound we need a more sophisticated mechanism of sharing such that *all* parties can contribute valid shares of each label, or someone is unanimously identified as a cheater. In Sect. 3.1 we construct exactly such a primitive, which we call one-or-nothing secret sharing with intermediaries (`1or0wi`). Intuitively, our one-or-nothing secret sharing with intermediaries achieves this goal by having each dealer use all of the parties as intermediaries to all share recipients; if sufficiently many intermediaries don't succeed in helping the dealer  $G$  give a share to a recipient  $P$ , then either the dealer or the recipient can be identified as corrupt, since they are in conflict with more than  $t$  intermediaries (we refer to Sect. 3.1 to a more detailed description of how this works).

We are now ready to describe our final protocol with identifiable abort for threshold  $3t < n$ . In the first round (which is over public peer-to-peer channels), the parties send their first-round messages of  $\Pi_{bc}$  along with the public keys produced by the key generation algorithm of `1or0wi`. In the second round (which is over broadcast), the parties execute the following steps:

1. They compute a garbling of the second-message function of  $\Pi_{bc}$ ;
2. they use `1or0wi` to share the labels of their garbled circuit;
3. they use `1or0wi` to vote for the labels of the garbled circuits of the other participants based on the first-round messages of  $\Pi_{bc}$  (received in the peer-to-peer round); and
4. they echo the first-round messages of  $\Pi_{bc}$  received in the first round.

Before computing the output, each party  $P_i$  performs some validations on the echoed messages. Namely,  $P_i$  checks that (a) all the parties generated their ballots for each garbled circuit based on the first-round messages that they echoed, and (b) all the parties have mutual successful communication with at least  $n - t$  others in the first round. If there is a party  $P_j$  that does not pass these checks, party  $P_i$  identifies  $P_j$  as a cheater. If all of the parties pass the checks, then party  $P_i$  invokes the `reconstruct` algorithm of `1or0wi`. If `reconstruct` blames party  $P_j$ ,  $P_i$  aborts and identifies that party as a cheater. Otherwise,  $P_i$  reconstructs labels for all the garbled circuits, uses the garbled circuits to obtain the second-round messages of  $\Pi_{bc}$ , and uses those second-round messages to complete the protocol and obtain the computation output.

Roughly speaking, the identifiable abort property is guaranteed since the one-or-nothing secret sharing with intermediaries is secure against active adversaries. Therefore, if the two validations (a) and (b) succeed, we can rely on the properties of `1or0wi` to guarantee that  $\Pi_{bc}$  is executed or a malicious party is identified.

More formally our protocol is described in Fig. 2 and we assume that the parties have access to the following tools:

**Tools.**

- A BC-BC, IA, CRS protocol i.e. a two-round broadcast protocol  $\Pi_{bc}$  achieving security with identifiable abort. (This could, for instance, be the protocol described by Cohen *et al.* [7].)  
 $\Pi_{bc}$  is represented by the set of functions  $\{\text{frst-msg}_i, \text{snd-msg}_i, \text{output}_i\}_{i \in [n]}$ .
- A garbling scheme ( $\text{garble}, \text{eval}, \text{simGC}$ ).
- A one-or-nothing secret sharing with intermediaries  
 $\text{1or0wi} = (\text{setup}, \text{keygen}, \text{share}, \text{vote}, \text{reconstruct})$  (defined in Sect. 3.1).

**Notation.** Let  $C_i(x_i, r_i, \text{msg}_1^1, \dots, \text{msg}_n^1)$  denote the boolean circuit that takes  $P_i$ 's input  $x_i$ , randomness  $r_i$  and the first round messages  $\text{msg}_1^1, \dots, \text{msg}_n^1$ , and outputs  $\text{msg}_i^2$ . For simplicity assume that  $(x_i, r_i)$  consists of  $z$  bits, and each first round message is  $\ell$  bits long, so each circuit has  $L = z + n \cdot \ell$  input bits. Note that  $C_i$  is public. Let  $g$  be the size of a garbled  $C_i$ .

Figure 2:  $\Pi_{2\text{pbc}}^{\text{id-abort}}$  with  $n > 3t$

**Private input.** Every party  $P_i$  has a private input  $x_i \in \{0, 1\}^*$  and randomness  $r_i \in \{0, 1\}^*$ .

**Setup.**

- CRS setup for one-or-nothing secret sharing with intermediaries:  
 $\text{crs} \leftarrow \text{setup}(1^\lambda)$ .
- Setup for  $\Pi_{bc}$  (which includes CRS when instantiated using the protocol of [7]).<sup>a</sup>

**First Round.** Each party  $P_i$  does the following:

1. Let  $(\text{sk}_i, \text{pk}_i) \leftarrow \text{keygen}(1^\lambda)$ , where  $\text{pk}_i = \{\text{pk}_i^{(1)} = (\text{pk}_i^{(1,1)}, \dots, \text{pk}_i^{(1,L)}), \dots, \text{pk}_i^{(n)} = (\text{pk}_i^{(n,1)}, \dots, \text{pk}_i^{(n,L)})\}$  is a vector of  $nL$  public keys with the corresponding vector of secret keys  $\text{sk}_i = \{\text{sk}_i^{(1)} = (\text{sk}_i^{(1,1)}, \dots, \text{sk}_i^{(1,L)}), \dots, \text{sk}_i^{(n)} = (\text{sk}_i^{(n,1)}, \dots, \text{sk}_i^{(n,L)})\}$  (We abuse notation slightly by assuming that  $\text{keygen}(1^\lambda)$  outputs a vector of public keys and secret keys; we do this for simplicity)
2. Let  $\text{msg}_i^1 \leftarrow \text{frst-msg}_i(x_i, r_i)$  be  $P_i$ 's first round message in  $\Pi_{bc}$ .
3. Send  $(\text{pk}_i, \text{msg}_i^1)$  to  $P_j$  for  $j \in [n]$ .

**Second Round.** Each party  $P_i$  does the following:

We specify multiple broadcast messages separately for clarity; however, they are all sent simultaneously as a single round of communication.

1. Let  $\text{pk}_{j \rightarrow i} = \{\text{pk}_{j \rightarrow i}^{(1)}, \dots, \text{pk}_{j \rightarrow i}^{(n)}\}$  denote the  $\text{pk}_j$  received privately from  $P_j$  ( $j \in [n]$ ), where  $\text{pk}_{j \rightarrow i}^{(k)} = (\text{pk}_{j \rightarrow i}^{(k,1)}, \dots, \text{pk}_{j \rightarrow i}^{(k,L)})$  for  $k \in [n]$ .
2. Compute  $(\text{GC}_i, \mathbf{K}_i) \leftarrow \text{garble}(1^\lambda, C_i; R_i)$ , where  $\mathbf{K}_i = \{K_{i,l}^{(0)}, K_{i,l}^{(1)}\}_{l \in [L]}$ .



3. For every  $l \in [z+1, \dots, L]$ , let  $s_{i,l} \leftarrow \text{share}(crs, \text{pk}_{1 \rightarrow i}^{(i,l)}, \dots, \text{pk}_{n \rightarrow i}^{(i,l)}, K_{i,l}^{(0)}, K_{i,l}^{(1)})$ . Broadcast  $\{s_{i,l}\}_{l \in [z+1, \dots, L]}$ .
4. Let  $(\nu_{i,z+1}, \dots, \nu_{i,L})$  denote the bits comprising  $(\text{msg}_{1 \rightarrow i}^1, \dots, \text{msg}_{n \rightarrow i}^1)$ , where  $\text{msg}_{j \rightarrow i}^1$  refers to  $\text{msg}_j^1$  received from  $P_j$  in Round 1.
5. For each  $k \in [n]$  and  $l \in [z+1, L]$ : Compute and broadcast  $\bar{s}_{i,l}^{(k)} \leftarrow \text{vote}(crs, \text{sk}_i^{(k,l)}, \text{pk}_{1 \rightarrow i}^{(k,l)}, \dots, \text{pk}_{n \rightarrow i}^{(k,l)}, \nu_{i,l})$ .  
Broadcast own garbled circuit:
6. Let  $(\nu_{i,1}, \dots, \nu_{i,z})$  denote the bits corresponding to  $(x_i, r_i)$ .
7. For  $l \in [z]$ , let  $K_{i,l} = K_{i,l}^{(\nu_{i,l})}$ .
8. Broadcast  $(\text{GC}_i, \{K_{i,l}\}_{l \in [z]})$ .  
Echo first-round messages:
9. Broadcast  $(\text{msg}_{1 \rightarrow i}^1, \dots, \text{msg}_{n \rightarrow i}^1)$ .  
Let  $\text{msg}_i^1 = \text{msg}_{i \rightarrow i}^1$  denote the party's own first-round message.

**Output Computation.** Each party  $P_i$  does the following:

If there is a party who did not generate ballots for each garbled circuit based on the first-round messages that she echoed, blame that party:

1. For  $j \in [n]$ : Check if  $\{\text{msg}_{k \rightarrow j}^1\}_{k \in [n]}$  broadcast by  $P_j$  is consistent with  $\{\bar{s}_{j,l}^{(k)}\}_{k \in [n], l \in [z+1, L]}$ <sup>b</sup>. Output **abort<sub>j</sub>** if the check fails. Else, set  $(\nu_{j,z+1}, \dots, \nu_{j,L})$  as the bits comprising  $(\text{msg}_{1 \rightarrow j}^1, \dots, \text{msg}_{n \rightarrow j}^1)$ .  
If there is a party who did not have mutual successful communication with at least  $n-t$  others in the first round, blame that party:
2. For  $j \in [n]$ : If there does not exist a set  $|S_j| \geq n-t$  such that, for  $k \in S_j$ ,  $\text{msg}_{j \rightarrow k}^1 = \text{msg}_j^1$  holds; output **abort<sub>j</sub>**.  
Decrypt the shares:
3. For  $k \in [n]$  (whose garbled circuit we will now consider):
  - (a) For  $l \in [z+1, L]$ , compute  $K_{k,l} \leftarrow \text{reconstruct}(crs, s_{k,l}, (\text{pk}_1^{(k,l)}, v_{1,l}, \bar{s}_{1,l}^{(k)}), \dots, (\text{pk}_n^{(k,l)}, v_{n,l}, \bar{s}_{n,l}^{(k)}))$ . If **reconstruct** returns  $\perp_{\text{id}}$ , output **abort<sub>id</sub>**. Else, continue.
  - (b) Evaluate  $\text{msg}_k^2 \leftarrow \text{eval}(\text{GC}_k, (K_{k,1}, \dots, K_{k,L}))$ . If the evaluation fails, output **abort<sub>k</sub>**.
4. Output  $y \leftarrow \text{output}_i(x_i, r_i, \text{msg}_1^1, \dots, \text{msg}_n^1, \text{msg}_1^2, \dots, \text{msg}_n^2)$ .

---

*P2P-BC, IA,  $t < \frac{n}{3}$  secure computation in the CRS model.*

<sup>a</sup> For simplicity (to avoid introducing additional notation), we assume implicitly that the set of functions  $\{\text{frst-msg}_i, \text{snd-msg}_i, \text{output}_i\}_{i \in [n]}$  of  $\Pi_{\text{bc}}$  use the relevant setup information.

<sup>b</sup> Note that in our construction of one-or-nothing secret sharing with intermediaries, it is possible to retrieve the corresponding vote directly from the ballot  $\bar{s}_{j,l}^{(k)}$ .

**Theorem 2 (P2P-BC, ID, CRS,  $n > 3t$ ).** *Let  $f$  be an efficiently computable  $n$ -party function and let  $n > 3t$ . Let  $\Pi_{bc}$  be a BC-BC, ID, CRS protocol that securely computes  $f$  with the additional constraint that the straight-line simulator can extract inputs from corrupt parties' first-round messages. Assuming that (garble, eval, simGC) is a secure garbling scheme, and (setup, keygen, share, vote, reconstruct) is a secure one-or-nothing secret sharing with intermediaries. Then,  $\Pi_{p2pbc}^{\text{id-abort}}$  securely computes  $f$  with identifiable abort over two rounds, the first of which is over peer-to-peer channels, and the second of which is over a broadcast and peer-to-peer channels.*

### 3.3 Feasibility Results for SIA

Our positive results for SIA rely on the following theorem (we defer its proof to the full version [10]).

**Theorem 3.** *Let  $\Pi_{bc}$  be a BC-BC protocol (respectively a P2P-BC) that securely computes  $f$  with identifiable abort security against  $t$  corruptions with the additional properties that the simulator can extract inputs from the first-round messages and it is efficient to check whether a given second-round message is correct. Then  $\Pi_{bc}$  securely computes  $f$  with selective identifiable-abort security against  $t$  corruptions when the second round is run over peer-to-peer channels instead.*

## 4 Lower Bounds

Our impossibility results for the setting where the first-round is over private peer-to-peer channels appear below. Our impossibility for the setting with public peer-to-peer channels in the first round appear in the full version [10].

**Theorem 4 (P2P-BC, SIA, CRS,  $n \leq 3t$ ).** *There exist functions  $f$  such that no  $n$ -party two-round protocol can compute  $f$  with selective identifiable abort against  $t \geq \frac{n}{3}$  corruptions while making use of broadcast only in the second round (i.e. where the first round is over peer-to-peer channels<sup>10</sup> and second round uses both broadcast and peer-to-peer channels).*

In our proof, we use the function  $f_{ot}$ . Let the input of  $P_1, P_2$  be a pair of strings  $x_1 = (z_0, z_1)$ ,  $x_2 = (z'_0, z'_1)$  where  $z_0, z_1, z'_0, z'_1 \in \{0, 1\}^\lambda$ , and the input of  $P_n$  be a choice bit  $x_n = c \in \{0, 1\}$ . The input of other parties is  $\perp$  (i.e.  $x_i = \perp$  for  $i \in [n] \setminus \{1, 2, n\}$ ).  $f_{ot}$  allows everyone to learn  $(z_c, z'_c)$ .

*Proof.* We prove Theorem 4 by contradiction. Let  $\Pi$  be an  $n$ -party protocol computing  $f_{ot}$  that achieves identifiable abort against  $t \geq \frac{n}{3}$  corruptions by using broadcast in the second round only.

For simplicity, we assume  $n = 3$  and  $t = 1$ . We analyze the following scenarios in an execution of  $\Pi$ .

<sup>10</sup> The peer-to-peer channels can be private or “open”.

**Scenario 1:** The adversary does the following on behalf of  $P_3$ .

**Round 1.** Compute and send messages based on input  $x_3 = 0$  and  $x_3 = 1$  to  $P_1$  and  $P_2$  respectively. (It is possible for the adversary to send inconsistent first-round messages as the first round is communicated over peer-to-peer channels.)

**Round 2.** Discard the first-round message from  $P_2$  and send messages based on input  $x_3 = 0$ . In other words,  $P_3$  pretends as if she behaved honestly using input  $x_3 = 0$  and did not receive a peer-to-peer message from  $P_2$  in the first round.

**Scenario 2:** Consider an adversary who corrupts  $P_2$ . Suppose the input of honest  $P_3$  is  $x_3 = 0$ . The adversary behaves as follows on behalf of  $P_2$ :

**Round 1.** Behave honestly as per protocol specifications, except that the peer-to-peer message to  $P_3$  is not sent.

**Round 2.** Pretend to have received first round messages from  $P_3$  based on  $x_3 = 1$ . In more detail, the adversary drops the first round peer-to-peer message received from  $P_3$  and replaces it by locally computing  $P_3$ 's first round message based on input  $x_3 = 1$  and some randomness (that the adversary can sample locally on behalf of  $P_3$ ). Note that the adversary can do this without being caught, due to the absence of PKI or correlated randomness.

*Claim.*  $\Pi$  is such that  $P_1$  in Scenario 1 learns the output  $(z_0, z'_0)$  with all but negligible probability.

*Proof.* First, we observe that the view of honest  $P_1$  in Scenario 1 is distributed identically to her view in Scenario 2. This is because in both scenarios,  $P_1$  observes the following conflict between  $P_2$  and  $P_3$ :  $P_3$  claims to have not received the first-round peer-to-peer message from  $P_2$  while  $P_2$  claims to have received first-round peer-to-peer message from  $P_3$  based on  $x_3 = 1$ . Therefore, to satisfy the guarantees of SIA, it must hold that either  $P_1$  aborts in both scenarios or obtains the output in both scenarios. The former is not possible, since  $P_1$  would identify the same cheater in both scenarios, which means that she would identify an honest party in one of the two scenarios (as the corrupt party is different in the two scenarios). We can thus infer from selective identifiable abort security guarantee of  $\Pi$  that both the above scenarios result in  $P_1$  receiving an output, with all but negligible probability.

The output obtained by  $P_1$  in Scenario 2 must include  $z_0$  as it should be computed with respect to the input  $x_3 = 0$  of honest  $P_3$  and input  $(z_0, z_1)$  of honest  $P_1$ . Therefore, the output obtained by  $P_1$  in Scenario 1 should also include  $z_0$  (with all but negligible probability). Infact, we can argue that the output obtained by  $P_1$  in Scenario 1 should in fact be  $(z_0, z'_0)$  (with all but negligible probability) to be consistent with the ideal realization of  $f$ . This is because the simulator in Scenario 1 can induce an output comprising of  $z_0$  only by invoking the ideal functionality with  $x_3 = 0$  on behalf of corrupt  $P_3$ , which fixes the output of  $P_1$  to include  $z'_0$  as per the definition of  $f$ .

We can thus conclude that the output obtained by honest  $P_1$  in Scenario 1 must be  $(z_0, z'_0)$ . Next, we consider another Scenario, say **Scenario 3** –

**Scenario 3:** Adversary corrupts  $P_1$  but behaves honestly throughout the protocol. Suppose the input of honest  $P_3$  is  $x_3 = 1$ .

First, it follows from the correctness of the protocol that since all parties including the corrupt parties behaved honestly in Scenario 3, the output computed must be in fact  $(z_1, z'_1)$  computed on honest inputs, which is obtained by all (including the adversary). Next, we show an attack by the adversary controlling  $P_1$  that allows her to obtain  $z'_0$  as well, which violates security (as an adversary corrupting  $P_1$  is not allowed to learn both inputs of honest  $P_2$  i.e.  $z'_0$  and  $z'_1$ , as per the ideal computation of  $f$ ). The main idea is that the adversary simulates *in her head* Scenario 1, where there was a conflict between  $P_3$  and  $P_2$ .

In the above execution of Scenario 3, let  $m_{i \rightarrow j}$  denotes the peer-to-peer first-round message sent by  $P_i$  to  $P_j$  and  $b_i$  denotes the second-round broadcast message sent by  $P_i$  (it is without loss of generality to assume that the second-round messages are over broadcast; since private communication in the second round can be realized by exchanging public keys in the first round).

**Round 1:** On behalf of  $P_3$ , the adversary chooses input  $x_3 = 0$  and some chosen randomness, say  $r_3$ . Using these values, the adversary recomputes the outgoing first-round peer-to-peer message from  $P_3$  to  $P_1$ , say  $\overline{m_{3 \rightarrow 1}}$ . However, the other first-round peer-to-peer messages i.e.  $m_{3 \rightarrow 2}, m_{2 \rightarrow 3}, m_{2 \rightarrow 1}, m_{1 \rightarrow 2}$  and  $m_{1 \rightarrow 3}$  are fixed to be the same as what were received during the execution of Scenario 3.

**Round 2:** Next, the adversary recomputes the second-round broadcast message of  $P_3$ , say  $\overline{b_3}$  as follows: Compute the broadcast message based on protocol specifications when  $P_3$  did not receive any first-round peer-to-peer message from  $P_2$ . Note that this message can be computed using input  $x_3 = 0$ , randomness  $r_3$  and the first-round peer-to-peer message  $m_{1 \rightarrow 3}$  received by  $\overline{P_3}$  from  $P_1$  (which the adversary knows). The broadcast message of  $P_1$ , say  $\overline{b_1}$  is recomputed based on honest input and randomness of  $P_1$ , the above simulated first-round peer-to-peer message  $\overline{m_{3 \rightarrow 1}}$  and  $m_{2 \rightarrow 1}$ . Lastly, the broadcast message of  $P_2$  is fixed to  $b_2$  (same as received in the execution).

We observe that the above simulation in her head, allows the adversary to obtain a view that is identically distributed to the view of honest  $P_1$  in Scenario 1. This is because both the simulation as well as Scenario 1 involve the messages  $\overline{m_{3 \rightarrow 1}}, \overline{b_1}$  and  $\overline{b_3}$  being based on  $x_3 = 0$ . We thus infer that the adversary should be able to compute the output of Scenario 1 as well.

Since the output of Scenario 1 is  $(z_0, z'_0)$ , we can conclude that the adversary of Scenario 3 learns both  $z'_0$  (via the simulation in her head) and  $z'_1$  (via the output of the execution) which violates security, since this is not allowed as per the ideal computation of  $f$ .

Lastly, we note that the above proof can be extended to  $n \leq 3t$  using player partitioning technique (An  $n$ -party protocol  $\Pi'$  tolerating  $t \geq n/3$  corruptions can be transformed into a 3-party protocol  $\Pi$  tolerating 1 corruption, by making a party in  $\Pi$  emulate the protocol steps of  $t$  parties in  $\Pi'$ ).

**Theorem 5 (BC-P2P, UA, CRS,  $t > 1$ ).** *There exist functions  $f$  such that no  $n$ -party two-round protocol can compute  $f$  with unanimous abort against  $t > 1$  corruptions while making use of broadcast only in the first round (i.e. where the first round uses both broadcast and peer-to-peer channels<sup>10</sup> and second round uses only peer-to-peer channels).*

The proof appears in the full version [10]

## References

1. Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Two round information-theoretic MPC with malicious security. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 532–561. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17656-3\\_19](https://doi.org/10.1007/978-3-030-17656-3_19)
2. Badrinarayanan, S., Miao, P., Mukherjee, P., Ravi, D.: On the round complexity of fully secure solitary mpc with honest majority. Cryptology ePrint Archive, Report 2021/241 (2021). <https://eprint.iacr.org/2021/241>
3. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 134–153. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-34961-4\\_10](https://doi.org/10.1007/978-3-642-34961-4_10)
4. Benhamouda, F., Lin, H.:  $k$ -round multiparty computation from  $k$ -round oblivious transfer via garbled interactive circuits. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 500–532. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-78375-8\\_17](https://doi.org/10.1007/978-3-319-78375-8_17)
5. Chen, M., Cohen, R., Doerner, J., Kondi, Y., Lee, E., Rosefield, S., Shelat, A.: Multiparty generation of an RSA modulus. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 64–93. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-56877-1\\_3](https://doi.org/10.1007/978-3-030-56877-1_3)
6. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: 18th Annual ACM Symposium on Theory of Computing, pp. 364–369. ACM Press, Berkeley, CA, USA, 28–30 May 1986. <https://doi.org/10.1145/12130.12168>
7. Cohen, R., Garay, J., Zikas, V.: Broadcast-optimal two-round MPC. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 828–858. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45724-2\\_28](https://doi.org/10.1007/978-3-030-45724-2_28)
8. Cohen, R., Lindell, Y.: Fairness versus guaranteed output delivery in secure multiparty computation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 466–485. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-662-45608-8\\_25](https://doi.org/10.1007/978-3-662-45608-8_25)
9. Damgård, I., Magri, B., Ravi, D., Siniscalchi, L., Yakoubov, S.: Broadcast-optimal two round MPC with an honest majority. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 155–184. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84245-1\\_6](https://doi.org/10.1007/978-3-030-84245-1_6)
10. Damgård, I., Ravi, D., Siniscalchi, L., Yakoubov, S.: Minimizing setup in broadcast-optimal two round MPC. Cryptology ePrint Archive, Report 2021/241 (2022). <https://eprint.iacr.org/2022/293>

11. Ganesh, C., Patra, A.: Broadcast extensions with optimal communication and round complexity. In: Giakkoupis, G. (ed.) 35th ACM Symposium Annual on Principles of Distributed Computing. pp. 371–380. Association for Computing Machinery, Chicago, IL, USA, 25–28 July 2016. <https://doi.org/10.1145/2933057.2933082>
12. Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 468–499. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-78375-8\\_16](https://doi.org/10.1007/978-3-319-78375-8_16)
13. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: On 2-round secure multiparty computation. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 178–193. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-45708-9\\_12](https://doi.org/10.1007/3-540-45708-9_12)
14. Goel, A., Jain, A., Prabhakaran, M., Raghunath, R.: On communication models and best-achievable security in two-round MPC. In: TCC, p. 690 (2021)
15. Dov Gordon, S., Liu, F.-H., Shi, E.: Constant-round MPC with fairness and guarantee of output delivery. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 63–82. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48000-7\\_4](https://doi.org/10.1007/978-3-662-48000-7_4)
16. Dov Gordon, S., Liu, F.-H., Shi, E.: Constant-round MPC with fairness and guarantee of output delivery. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 63–82. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48000-7\\_4](https://doi.org/10.1007/978-3-662-48000-7_4)
17. Ishai, Y., Kumaresan, R., Kushilevitz, E., Paskin-Cherniavsky, A.: Secure computation with minimal interaction, revisited. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 359–378. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48000-7\\_18](https://doi.org/10.1007/978-3-662-48000-7_18)
18. Ishai, Y., Kushilevitz, E., Paskin, A.: Secure multiparty computation with minimal interaction. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 577–594. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-14623-7\\_31](https://doi.org/10.1007/978-3-642-14623-7_31)
19. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. *ACM Trans. Program. Lang. Syst.* 4(3), 382–401 (1982)
20. Patra, A., Ravi, D.: On the exact round complexity of secure three-party computation. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 425–458. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-96881-0\\_15](https://doi.org/10.1007/978-3-319-96881-0_15)



# Sublinear-Communication Secure Multiparty Computation Does Not Require FHE

Elette Boyle<sup>1,2</sup>, Geoffroy Couteau<sup>3</sup>, and Pierre Meyer<sup>1,3</sup>(✉)

<sup>1</sup> Reichman University, Herzliya, Israel  
eboyle@alum.mit.edu

<sup>2</sup> NTT Research, Sunnyvale, USA

<sup>3</sup> Université Paris Cité, CNRS, IRIF, Paris, France  
{couteau,pierre.meyer}@irif.fr

**Abstract.** Secure computation enables mutually distrusting parties to jointly compute a function on their secret inputs, while revealing nothing beyond the function output. A long-running challenge is understanding the required communication complexity of such protocols—in particular, when communication can be *sublinear* in the circuit representation size of the desired function. Significant advances have been made affirmatively answering this question within the *two-party* setting, based on a variety of structures and hardness assumptions. In contrast, in the *multi-party* setting, only one general approach is known: using Fully Homomorphic Encryption (FHE). This remains the state of affairs even for just three parties, with two corruptions.

We present a framework for achieving secure sublinear-communication  $(N + 1)$ -party computation, building from a particular form of Function Secret Sharing for only  $N$  parties. In turn, we demonstrate implications to sublinear secure computation for various function classes in the 3-party and 5-party settings based on an assortment of assumptions not known to imply FHE.

**Keywords:** Foundations · Secure Multiparty Computation · Function Secret Sharing · Private Information Retrieval

## 1 Introduction

Secure computation enables mutually distrusting parties to jointly compute a function on their secret inputs, while revealing nothing beyond the function output. Since the seminal feasibility results of the 1980s [6, 18, 29, 41], a major challenge in the area has been if and when it is possible to break the “circuit-size barrier.” This barrier refers to the fact that all classical techniques for secure computation required a larger amount of communication than the size of a boolean

---

**Supplementary Information** The online version contains supplementary material available at [https://doi.org/10.1007/978-3-031-30617-4\\_6](https://doi.org/10.1007/978-3-031-30617-4_6).

circuit representing the function to be computed. In contrast, insecure computation only requires exchanging the inputs, which are usually considerably smaller than the entire circuit.

This challenge eluded the field for nearly two decades, aside from partial results that either required exponential computation [4,36], or were limited to very simple functions (such as point functions [20,21,35] or constant-depth circuits [3]). This changed with the breakthrough result of Gentry [27] on *fully homomorphic encryption* (FHE). FHE is a powerful primitive supporting computation on encrypted data, which can be used to build asymptotically optimal-communication protocols in the computational setting [2,24].

In the years after, significant progress has been made toward broadening the set of techniques and class of assumptions under which sublinear-communication secure computation can be built. A notable such approach is via *homomorphic secret sharing* (HSS) [11]. HSS can be viewed as a relaxation of FHE, where homomorphic evaluation can be distributed among two parties who do not interact with each other, but which still suffices for low-communication secure computation. Following this approach (explicitly, building forms of HSS for  $NC^1$ ), sublinear-communication secure protocols have been developed based on the Decisional Diffie-Hellman (DDH) assumption [11], Decision Composite Residuosity (DCR) [26,37,40], and further algebraic structures, including a class of assumptions based on class groups of imaginary quadratic fields [1]. It was extended to a flavor of the Learning Parity with Noise (LPN) assumption (via HSS for log log-depth circuits) by [23]. Othogonally to these approaches, which rely on computational assumptions, [22] built sublinear-communication secure computation under an assumption of correlated randomness.

Very recently, a work of [9] demonstrated an alternative approach to sublinear secure computation through a certain form of rate-1 batch oblivious transfer (OT), resulting in protocols based on a weaker form of LPN plus Quadratic Residuosity.

However, aside from the original approach via FHE, *all* of the above techniques are strongly tied to the *two-party* setting, as opposed to the general setting of multiple parties, where all but one can be corrupt.

More concretely, while  $N$ -party HSS with security against  $(N - 1)$  colluding parties would directly imply the desired result, actually achieving such a primitive for rich function classes (without tools already implying FHE) beyond  $N = 2$ , is a notable open challenge in the field. The 2-party setting provides special properties leveraged within HSS constructions; e.g., given an additive secret sharing of 0, it implies the two parties hold identical values. These properties completely break down as soon as one steps to three parties with security against two. This separation can already be showcased for very simple function classes, such as HSS for equality test (equivalently, “distributed point functions” [10,28]), where to this date an exponential gap remains between the best constructions in the 2-party versus 3-party setting [10]. For  $N \geq 3$ , there are constructions of  $N$ -party FSS for all polynomial-time computable functions, but only from LWE, by using *additive-function-sharing* spooky encryption (AFS-spooky encryption) [25], or from subexponentially secure *indistinguishability obfuscation* [10].



Additionally, [14] turns this FSS from spooky encryption into additive HSS. In addition, approaches from the 2-party batch OT primitive seem also to be strongly tied to two parties.

Despite great progress in the two-party setting—and the fundamental nature of the question—to date, sublinear secure computation results for 3 or more parties remain stuck in the “2009 era”: known only for very simple functions (e.g., constant-degree computations), or based on (leveled) FHE.

## 1.1 Our Results

We present a new framework for achieving secure computation with low communication. Ultimately our approach yields new sublinear secure computation protocols for various circuit classes in the 3-party and 5-party settings, based on an assortment of assumptions not known to imply FHE.

*General Framework.* Our high-level approach centers around Function Secret Sharing (FSS) [10], a form of secret sharing where the secret object and shares comprise succinct representations of *functions*. More concretely, FSS for function class  $\mathcal{F}$  allows a client to split a secret function  $f \in \mathcal{F}$  into function shares  $f_1, \dots, f_N$  such that any strict subset of  $f_i$ 's hide  $f$ , and for every input  $x$  in the domain of  $f$  it holds that  $\sum_{i=1}^N f_i(x) = f(x)$ . (This can be seen as the syntactic dual of HSS, where the role of input and function are reversed; we refer the reader to e.g. [14] for discussion.<sup>1</sup>)  $N$ -party FSS/HSS for sufficiently rich function classes is known to support low-communication  $N$ -party secure computation, but lack of multi-party FSS constructions effectively leaves us stuck at  $N = 2$ .

The core conceptual contribution of this work is the following simple framework, which enables us to achieve  $(N + 1)$ -party secure computation by using a form of FSS for only  $N$  parties.

**Proposition 1 (( $N + 1$ )-PC from  $N$ -FSS framework, informal).** *For any ensemble of polynomial-size circuits  $\mathcal{C} = \{C_\lambda\}$ , consider an  $N$ -party FSS scheme for the class of “partial evaluation” functions  $\{C_\lambda(\cdot, x_1, \dots, x_N)\}_{\lambda, x_1, \dots, x_N}$ , and define the following sub-computation functionalities:*

- $\mathcal{F}_{\text{SD}}^{\text{FSS}}$ :  $N$ -party secure FSS share distribution, where each party  $P_i$  holds input  $x_i$  (and  $\lambda$ ), and learns the  $i$ th FSS key  $f_i$  for the function  $C_\lambda(\cdot, x_1, \dots, x_N)$ .
- $\mathcal{F}_{\text{OE}}^{\text{FSS}}$ : Two-party oblivious FSS evaluation, where party  $P_i$  holds an FSS key  $f_i$ , party  $P_0$  holds input  $x_0$ , and  $P_0$  learns the  $i$ th output  $f_i(x_0)$ .

*Then there exists a  $(N + 1)$ -party protocol for securely computing  $\mathcal{C}$  making one call to  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  and  $N$  calls to  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$ .*

Once expressed in this form, the resulting  $(N + 1)$ -party protocol becomes an exercise: Roughly, it begins by having parties  $1, \dots, N$  jointly execute  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  on their inputs  $x_1, \dots, x_N$  to each receive a function share  $f_i$  of the secret function  $f(x_0) := C_\lambda(x_0, x_1, \dots, x_N)$ , and then each run a pairwise execution of  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$

<sup>1</sup> Indeed, we will refer to both notions, using each when more conceptually convenient.

together with the remaining party  $P_0$  in order to obliviously communicate the  $i$ th output share  $f_i(x)$ . Given these shares,  $P_0$  can compute the final output as  $\sum_{i=1}^N f_i(x_0)$ . (See the Technical Overview for more detailed discussion.)

The communication of the resulting protocol will be dominated by the executions of  $\mathcal{F}_{\text{SD}}^{\text{FSS}}, \mathcal{F}_{\text{OE}}^{\text{FSS}}$ . Of course, the technical challenge thus becomes if and how one can construct corresponding FSS schemes which admit secure share distribution and oblivious evaluation with *low communication*.

*Instantiating the Framework.* We demonstrate how to instantiate the above framework building from known constructions of Homomorphic Secret Sharing (HSS) combined with a version of low-communication PIR.

We first identify a structural property of an FSS scheme which, if satisfied, then yields a low-communication procedure for oblivious share evaluation, through use of a certain notion of “correlated” (batch) Symmetric Private Information Retrieval (SPIR). Loosely, correlated SPIR corresponds to a primitive where a client wishes to make *correlated* queries into  $m$  distinct size- $S$  databases held by a single server. Without correlation between queries, the best-known PIR constructions would require  $m \cdot \text{polylog}(S)$  communication. However, it was shown in [9] that if the  $m$  index queries (each  $\log S$  bits) are given by various subsets of a fixed bit string of length  $n \ll m \log S$  held by the client, then (using the rate-1 batch OT constructions from [17]) this batch SPIR can be performed with significantly lower communication.

We then demonstrate that FSS schemes with the necessary structural property can be realized from existing constructions of HSS. Loosely speaking, the FSS evaluation procedure will be expressible as a polynomial (which depends on  $x_1, \dots, x_N$ ) evaluated on the final input  $x_0$ , and the HSS will enable the  $N$  parties to compute additive secret shares of the coefficients of this corresponding polynomial.

We further extend the approach to support an underlying HSS scheme satisfying only a weaker notion of correctness, with *inverse-polynomial* (Las Vegas) *error*. In such scheme, homomorphic evaluation may fail with noticeable probability (over the randomness of share generation), in a manner identifiable to one or more parties. This is the notion satisfied by the 2-party HSS constructions from Decisional Diffie-Hellman [11], or Learning With Errors with only a polynomial-size modulus [16, 25]. This error must be removed in our construction while incurring minimal additional interaction. We demonstrate how to do so, using (standard) Private Information Retrieval [21] and punctured pseudorandom functions [8, 15, 33]. Note that the former is implied by correlated SPIR, and the latter implied by any one-way function, so that these tools do not impose additional assumptions in the statement below.

**Theorem 2 (Sublinear MPC, informal).** *For any ensemble of polynomial-size circuits  $\mathcal{C} = \{\mathcal{C}_\lambda\}$  of size  $s$ , depth  $\log \log s$ , and with  $n$  inputs and  $m$  outputs, if there exists the following:*

- *Correlated Symmetric Batch PIR, for  $m$  size- $s$  databases where queries come from  $n$  bits, with communication  $O(n + m + \text{poly}(\lambda) + \text{comm}(s))$  for some function  $\text{comm}$ .*

- (Las Vegas)  $N$ -party Homomorphic Secret Sharing with compact shares (size  $O(n)$  for input size  $n$ ), for the class of  $\log$ -depth boolean circuits.

Then there exists a secure  $(N+1)$ -party computation protocol for  $\mathcal{C}$  with communication  $O(n+m+\text{poly}(\lambda)+N \cdot \text{comm}(s))$ . In particular, sublinearity is achieved when  $N \cdot \text{comm}(s) \in o(s)$ .

*Remark 3 (Compiling Sublinear MPC from Passive to Active Security).* In this work, we focus on security against semi-honest adversaries. However, all our results extend immediately to the malicious setting, using known techniques. Indeed, to get malicious security while preserving sublinearity, one can just use the seminal GMW compiler [29] with zero-knowledge arguments, instantiating the ZKA with (interactive) succinct arguments [36]. Using Kilian’s PCP-based 4-move argument [34], which has polylogarithmic communication, this can be done using any collision-resistant hash function. The latter are implied by all assumptions under which we base sublinear MPC, hence our results generalise directly to the malicious setting. This observation was made in previous works on sublinear-communication secure computation (e.g. [9, 11, 23]).

*Remark 4 (Beyond Boolean circuits).* The above approach can be extended to arithmetic circuits over general fields  $\mathbb{F}$ , by replacing the correlated SPIR with an analogous form of (low-communication) *correlated oblivious polynomial evaluation* (OPE). We discuss and prove this more general result in the main body, but focus here on the Boolean setting, as required instantiations of such correlated-OPE beyond constant-size fields are not yet currently known.

*Resulting Constructions.* Finally, we turn to the literature to identify constructions of the required sub-tools, yielding resulting sublinear secure computation results from various mathematical structures and computational assumptions.

**Corollary 5 (Instantiating the framework, informal).** *There exists secure 3-party computation for evaluating Boolean circuits of size  $s$  and depth  $\log \log s$  with  $n$  inputs and  $m$  outputs, with communication complexity  $O(n+m+\sqrt{s} \cdot \text{poly}(\lambda) \cdot (n+m)^{2/3})$  based on the Learning Parity with Noise (LPN) assumption for any inverse-polynomial error rate, together with any of the following additional computational assumptions:*

- Decisional Diffie-Hellman (DDH)
- Learning with Errors with polynomial-size modulus (poly-modulus LWE)
- Quadratic Residuosity (QR) + superpolynomial LPN<sup>2</sup>

*This can be extended under the same assumptions to secure 3-party computation of general “layered” (in fact, only locally synchronous<sup>3</sup>) circuits of depth  $d$*

<sup>2</sup> Superpolynomial hardness of LPN with a small inverse-superpolynomial error rate, but few samples, as assumed in [23].

<sup>3</sup> A circuit is *layered* [31] if all gates and inputs are arranged into layers, such that any wire only connects one layer to the next, but each input may occur multiple times at different layers. A layered circuit is *locally synchronous* [5] if each input occurs exactly once (but at an arbitrary layer). A locally synchronous circuit is *synchronous* [32] if all inputs are in the first layer.

and size  $s$  with communication  $O(s/\log \log s + d^{1/3} \cdot s^{2(1+\epsilon)/3} \cdot \text{poly}(\lambda))$ , for arbitrary small constant  $\epsilon$ . The latter is sublinear in  $s$  whenever  $d = o(s^{1-\epsilon}/\text{poly}(\lambda))$ , i.e., the circuit is not too “tall and skinny.”

If we further assume the existence of a constant-locality PRG with some polynomial stretch and the super-polynomial security of the Decisional Composite Residuosity (DCR) assumption, then the above extends to the 5-party setting, both for loglog-depth boolean circuits and for layered boolean circuits.

More concretely, the required notion of correlated SPIR was achieved in [9], building on [17], from a selection of different assumptions. The required HSS follows for  $N = 2$  from DDH from [11], LWE with polynomial-size modulus from [16, 25], DCR from [37, 40], and from superpolynomial LPN from [23]. It holds for  $N = 4$  from DCR from [19] (with some extra work, complexity leveraging, and restrictions; see technical section). Note that combining the works of [17, 37] seems to implicitly yield rate-1 batch OT from DCR, and in turn correlated SPIR [9]: if true, the assumptions for sublinear-communication five-party MPC can be simplified to constant-locality PRG, LPN, and superpolynomial DCR (without the need for DDH, LWE, or QR). Since this claim was never made formally, we do not use it.

A beneficial consequence of our framework is that future developments within these areas can directly be plugged in to yield corresponding new constructions and feasibilities.

## 1.2 Technical Overview

*General Framework.* Recall the secure computation framework via homomorphic secret sharing (HSS). Given access to an  $N$ -party HSS scheme supporting homomorphic evaluation of the desired circuit  $C$ , the parties begin by jointly HSS-sharing their inputs via a small secure computation. Each party can then homomorphically evaluate the circuit  $C$  on its respective HSS share without interaction, resulting in a short output share that it exchanges with all other parties. The parties can then each reconstruct the desired output by combining the evaluated shares (for standard HSS, this operation is simply addition). The resulting MPC communication cost scales only with the complexity of HSS share generation plus exchange of (short) output shares, but remains otherwise independent of the complexity of  $C$ .

In theory, this approach provides sublinear secure computation protocols for any number of parties  $N$ . In practice, however, we simply do not have HSS constructions for rich function classes beyond  $N = 2$  with security against collusion of two or more corrupted parties, crucial for providing the corresponding MPC security. This remains a standing open question that has received notable attention, and unfortunately seems to be a challenging task.

A natural question is whether the above framework can somehow be modified to extend beyond the number of parties  $N$  supported by the HSS, for example to  $N' = N + 1$ . The issue with the above approach is that parties cannot afford to secret share their input to any  $N$ -subset in which they do not participate, as

all parties within this subset may be corrupt, in which case combining all HSS shares reveals the shared secrets.

Instead, suppose that only the  $N$  parties share their inputs amongst each other. In this case, there is no problem with all  $N$  shareholding parties being corrupt, as this reveals only their own set of inputs. But, we now have a challenge: how to involve the final party's input into the computation?

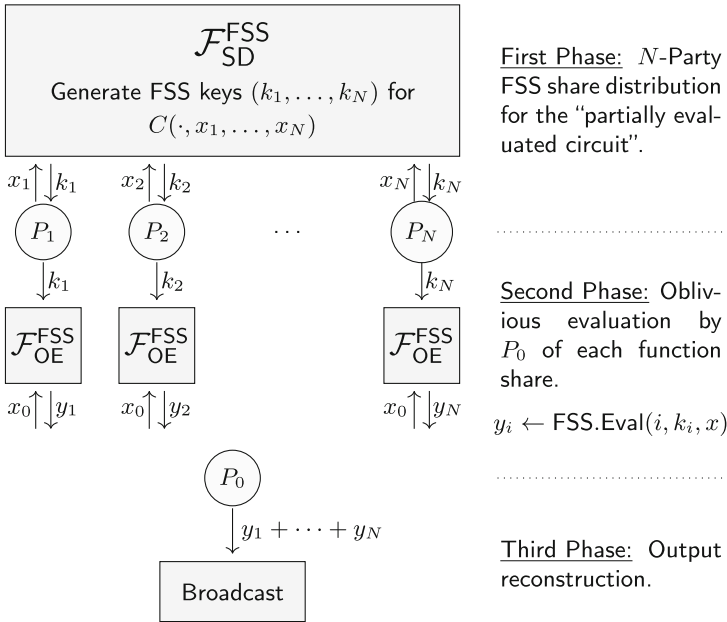
In the HSS framework, parties each homomorphically evaluated the public  $C$  on shares. Suppose, on the other hand, the HSS supports homomorphic evaluation of the *class* of functions  $C_{x_0} := C(x_0, \cdot, \dots, \cdot)$ . Or, more naturally, consider a dual view: Where the  $N$  parties collectively generate shares of a secret *function*  $C(\cdot, x_1, \dots, x_N)$  with their inputs hardcoded, which accepts a single input  $x_0$  and outputs  $C(x_0, \dots, x_N)$ . That is, using *function* secret sharing (FSS).

Of course, normally in FSS we think of the input on which the function is to be evaluated (in this case,  $x_0$ ) as a public value, which each shareholder will know. Here, this clearly cannot be the case. Instead, we consider a modified approach, where each of the  $N$  FSS shareholders will perform a pairwise *oblivious evaluation* procedure, with the final  $(N + 1)$ st party  $P_0$ . That is, the  $i$ th shareholder holds the  $i$ th function key FSS  $k_i$ , which defines a share evaluation function “ $f_i$ ” =  $\text{FSS.Eval}(i, k_i, \cdot)$ . As a result of the oblivious evaluation, party  $P_0$  will learn the evaluation  $y_i = \text{FSS.Eval}(i, k_i, x_0)$  of this function on its secret input  $x_0$ , and neither party will learn anything beyond this; in particular,  $P_0$  does not learn  $k_i$ , and  $P_i$  does not learn  $x_0$ . At the conclusion of this phase, party  $P_0$  learns exactly the set of  $N$  output shares, and can reconstruct the final output  $C(x_0, \dots, x_N) = y_1 + \dots + y_N$  and send to all parties.

The corresponding high-level protocol template is depicted in Fig. 1. Here,  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  represents an ideal  $N$ -party functionality for  $N$ -FSS share generation (defined formally in Fig. 2 of Sect. 3), where each party provides its input  $x_i$  and receives its FSS share  $k_i$ .  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$  represents an ideal two-party functionality for oblivious FSS share evaluation (defined formally in Fig. 3 of Sect. 3), where  $P_i$  and  $P_0$  respectively provide inputs  $k_i$  and  $x_0$ , and  $P_0$  learns the evaluation  $\text{FSS.Eval}(i, k_i, x_0)$ .

Consider the (passive) security of the proposed scheme against up to  $N$  corruptions. If the corrupted parties are (any subset of) those holding FSS shares, then since the parties execute a secure computation for share generation, their view is restricted to a subset of FSS key shares  $(k_i)_{i \in T}$ , which hides any honest parties' inputs  $(x_i)_{i \in [N] \setminus T}$  by the security of the FSS. (Note if all  $N$  shareholding parties are corrupt, then this statement holds vacuously, as no honest parties' inputs were involved.) If the corrupted parties include  $P_0$  together with a (necessarily *strict*) subset of FSS shareholders, then their collective view consists of a strict subset of FSS keys  $(k_i)_{i \in T}$  together with evaluated output shares  $(y_i)_{i \in [N]}$ . However, the security of the FSS combined with the additive reconstruction of output shares implies this reveals nothing beyond the function output.<sup>4</sup>

<sup>4</sup> Note that in fact we do not need FSS with *additive* reconstruction, but rather any form of reconstruction will suffice, as long as the output shares provide this property of revealing nothing beyond the function output. We formalize this property, and prove it holds for additive reconstruction, in the full version of the paper.



**Fig. 1.** Template for  $(N + 1)$ -party sublinear secure computation of  $C$  from  $N$ -party additive FSS.

Now, in order for this framework to provide low communication, it must be the case that we have an FSS scheme for the relevant partial-evaluation function class  $\{f_{\alpha_1, \dots, \alpha_N} = C(\cdot, \alpha_1, \dots, \alpha_N)\}$ , for which the following two steps can be performed succinctly:

- Secure  $N$ -party FSS share generation, and
- Oblivious evaluation by  $P_0$  of each function share.

We next address approaches for how each of these pieces can be achieved.

*Oblivious Evaluation for “Loglog-depth” FSS via PIR.* Consider first the pairwise oblivious FSS evaluation procedure, where  $P_0$  holds  $x_0$ , party  $P_i$  holds FSS key  $k_i$ , and  $P_0$  should learn  $\text{FSS.Eval}(i, k_i, x_0)$ .

Since this is reduced to a 2-party functionality, a natural first place to look would be for FSS schemes where  $\text{FSS.Eval}(i, k_i, \cdot)$  is within a function class already admitting low-communication 2-party secure computation. Unfortunately, this is more challenging than it sounds. While sublinear-communication 2PC exists for general layered circuits from a variety assumptions, recall that the sublinearity will be here in the complexity not of  $C$ , but of  $\text{FSS.Eval}$ , almost certainly a more complex computation.

Indeed, the idea of increasing the number of parties by homomorphically evaluating an  $\text{HSS.Eval}$  has previously been considered in the related setting of HSS, and hit similar limitations. For example, relatively strong HSS schemes based

on DDH or DCR support homomorphic evaluation (and thus secure computation with very low communication) of  $NC^1$ ; but, the corresponding operations required to actually *compute*  $\text{HSS.Eval}$  itself lies outside of  $NC^1$ . In [13], this was addressed by instead securely computing a (low-depth) *randomized encoding* of the evaluation operation, effectively squashing the depth of the computation to be securely performed. This enabled them to achieve low round complexity, but resulted in large communication (scaling with the size of the entire  $\text{HSS.Eval}$  circuit). Recently, it was shown by Chillotti et al. [19] that for the specific DCR-based HSS construction of [37, 40],  $\text{HSS.Eval}$  for homomorphically evaluating a constant-degree computation can be computed within  $NC^1$ . However, this only gives low-communication secure computation for constant-degree functions, which will not suffice for overall sublinearity.

Instead, we take a different approach, going beyond black-box use of existing sublinear 2PC results. While the full  $\text{FSS.Eval}(i, k_i, x_0)$  computation itself may be complex, suppose it is the case that it can be decomposed into two parts: (1) some form of precomputation, depending only on  $i$  and  $k_i$ , followed by (2) computation on  $x_0$ , which is of low complexity. More concretely, consider the output of part (1) to be a new circuit  $C_{\text{Eval}}$  whose input is  $x_0$  and output is  $\text{FSS.Eval}(i, k_i, x_0)$ , and suppose it is the case that  $C_{\text{Eval}}$  has low  $\log \log(s)$  depth (where  $s$  is the size of the original circuit  $C$  the parties wish to compute in the MPC). Note that while  $C_{\text{Eval}}$  has low depth, its identity depends on the secret  $k_i$  (of  $P_i$ ), so that black-box secure computation of  $C_{\text{Eval}}$  does not apply.

On the other hand, opening the box of one such recent secure computation protocol, we identify that an intermediate tool developed actually has stronger implications. The tool is *correlated batch symmetric PIR*, for short correlated SPIR [9], which as discussed above, enables low-communication of several batched instances of (single-server) SPIR whose queries are correlated. In this case, the  $m$  “databases” will be defined implicitly by the  $m$  output bits of the circuit  $C_{\text{Eval}}$ . Because  $C_{\text{Eval}}$  is  $\log \log s$  depth as a function of its input  $x_0$  (and circuits are taken to be fan-in 2), each computed output bit depends on at most  $\log s$  bits of  $x_0$ , and as such can be represented as a size- $s$  database indexed by the corresponding  $\log s$  input bits. Oblivious evaluation of  $C_{\text{Eval}}$  on  $x_0$  can then be achieved by  $P_0$  making  $m$  batch queries into these databases, where the collective query bits are all derived from various bits of the single string  $x_0$ .

As a brief aside: Extending to larger arithmetic spaces, the role of correlated SPIR here can be replaced by an analogous version of correlated Oblivious Polynomial Evaluation (OPE). Here, a  $\log \log s$  depth arithmetic circuit  $C_{\text{Eval}}$  can be expressed as a secret multivariate polynomial in  $x_0$  of size  $\text{poly}(s)$ , where each monomial depends on at most  $\log s$  elements of the arithmetic vector  $x_0$ . Unfortunately, we are not presently aware of tools for achieving low-communication correlated OPE beyond constant-size fields. However, we include this in the technical exposition, in case such techniques are later developed. We note that the final steps in our instantiation (described in the following) do hold over larger arithmetic spaces under certain computational assumptions.

“Loglog-depth” FSS from HSS. Consider an ensemble  $\mathcal{C} = \{C_\lambda\}$  of Boolean circuits of size  $s$  and depth  $\log \log s$ . The remaining goal is to obtain FSS for the corresponding class of partial-evaluation functions  $\{C_\lambda(\cdot, x_1, \dots, x_N)\}$  for which the FSS evaluation  $C_{\text{Eval}}$  is  $(\log \log s)$ -depth, as discussed above.

From the structure of  $C_\lambda$ , the evaluation of  $C_\lambda(x_0, \dots, x_N)$  on all inputs can be expressed as a  $\text{poly}(s)$ -size multivariate polynomial in the bits  $x_i[j]$  of the  $x_i$ , where each monomial is of degree at most  $\log s$ . When viewed as a function of just  $x_0$ , we thus have  $\text{poly}(s)$ -many monomials in the bits of  $x_0$  whose coefficients  $p_j$  are each formed by the product of at most  $\log s$  bits from the inputs  $x_1, \dots, x_N$ . That is,  $\sum_j p_j \prod_{\ell \in S_j} x_0[\ell]$ , where each  $|S_j| \leq \log s$  is a publicly known set.

If the  $N$  parties can somehow produce *additive secret shares*  $\{p_j^{(i)}\}_{i \in [N]}$  of each one of these coefficients  $p_j$ , then this would constitute the desired FSS evaluation: Indeed, the  $i$ th share evaluation  $\text{FSS.Eval}(i, k_i, x_0)$  would be computable as  $y^{(i)} = \sum_j p_j^{(i)} \prod_{\ell \in S_j} x_0[\ell]$ , satisfying  $\sum_{i=1}^N y^{(i)} = \sum_j p_j \prod_{\ell \in S_j} x_0[\ell] = C_\lambda(x_0, \dots, x_N)$ . Further, each  $\text{FSS.Eval}(i, k_i, \cdot)$  is expressible as a  $(\log \log s)$ -depth circuit in  $x_0$ —as required from the previous discussion.

The question is how to *succinctly* reach a state where the  $N$  parties hold these coefficient secret shares. Of course, direct secure computation is not an option, as even the output size is large,  $\text{poly}(s)$ . However, this is not a general computation. Suppose we have access to an HSS scheme supporting homomorphic evaluation of  $\log \log s$  depth operations. Such constructions are known to exist from a variety of assumptions (as discussed after Corollary 5). Then, if the parties HSS share their respective inputs  $x_1, \dots, x_N$ , they can *locally* evaluate additive shares of the corresponding  $(\log s)$ -products  $p_j$ .

The corresponding  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  operation will thus correspond to the HSS.Share procedure of the HSS scheme on the parties’ collective inputs. If the HSS scheme has a compact sharing procedure, then this will be computable with sufficiently low communication. Note that vanilla usage of some HSS schemes will not provide the required compactness (e.g., including structured ciphertexts of the input bits); however, using standard hybrid encryption tricks this can be facilitated.

“Loglog-depth” FSS from Las Vegas HSS. An additional challenge arises, however, when the underlying HSS scheme we attempt to use provides correctness only up to *inverse-polynomial error*. This is the case, for example, in known 2-party HSS schemes for  $NC^1$  from DDH [11] or from LWE with polynomial-size modulus [16, 25]. In these schemes, the inverse-polynomial error rate  $\delta$  can be chosen as small as desired, but shows up detrimentally as  $1/\delta$  in other scheme parameters (runtime for the DDH scheme; modulus size for LWE).

This means with noticeable probability, the shares of at least one of the coefficients  $p_j$  from above will be computed incorrectly. Even worse, as typical in these settings, the parties cannot learn or reveal where errors truly occurred, as this information is dependent on the values of the secret inputs. This remains a problem even if the HSS scheme is “Las Vegas,” in the sense that for every error at least one of the parties will identify that a potential-error event has occurred (i.e., will evaluate output share as  $\perp$ ). Even then, the flagging party must not



learn whether an error truly took place, and the other party must not learn that a potential error was flagged.

We present a method for modifying the HSS-based FSS sharing procedure from above, to remove the error in the required homomorphic evaluations, while hiding from the necessary parties where these patches took place. We focus on the 2-party case, and further assume the HSS has a succinct protocol (communication linear in the input size, up to an additive  $\text{poly}(\lambda)$  term) for distributing the shares of the HSS, where homomorphic evaluation can take place across different sets of shared values. This is the case for known Las Vegas HSS schemes.

This procedure can be viewed as a modification to either the `Share` or `Eval` portion of the FSS. By viewing it as part of `FSS.Share`, we automatically fit into the framework of the previous sections. Namely, this can be viewed as a new `FSS.Share` (or  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$ ) procedure with relatively large computational complexity (comparable to the truth table of the shared function), but which we show *admits a low-communication secure computation procedure*. We describe the sharing procedure directly via the achieving protocol; the corresponding `FSS.Share` procedure can be inferred.

First, note that by taking the inverse-polynomial error rate  $\delta$  to be sufficiently small, we can guarantee with high probability that the total number of potential-error flags  $\perp$  obtained by any party is at most the security parameter,  $\lambda$ . The sharing protocol begins by HSS sharing the inputs  $(s_0, s_1) \leftarrow \text{HSS.Share}(x_1, x_2)$  as usual. Then, each party homomorphically evaluates all required values corresponding to shares of each of the coefficients  $p_j$ . For each party  $P_i$  ( $i \in \{1, 2\}$ ), denote these values in an array  $T_i$ , which contains at most  $\lambda$  positions in which  $T_i[j] = \perp$ . For each such position  $j^*$ ,  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  sets  $T_i[j^*] = 0$ , and must now “patch” the missing value. Consider this procedure for party  $P_1$  ( $P_2$  will be reversed).

In order to compute the correct output (i.e., coefficient  $p_j$ ) in this position, the parties run a small-scale secure protocol that HSS shares the *index* position  $j^*$  of each  $\perp$  symbol of  $P_1$ . This enables them to homomorphically re-evaluate shares of the corresponding coefficient term  $p_{j^*}$ , in a way that hides the index  $j^*$  from  $P_2$  (note that this computation, with index selection, remains within  $NC^1$ ). In fact, by re-evaluating this computation  $\lambda$ -many times, then with overwhelming probability, at least one is error-free. By running a small-scale secure computation on these shares, we can assume that the parties hold additive shares of the correct value  $p_{j^*}$ .

It would seem the remaining step is for  $P_1$  to somehow learn the correct value  $p_{j^*}$  offset by  $P_2$ 's share  $T_2[j^*]$ , while keeping  $j^*$  hidden from  $P_2$ . However, the situation is somewhat more sticky. The problem is that in the original HSS evaluation,  $P_1$  learns not only  $\perp$ , but also a candidate output share. By receiving the *correct* output share  $(p_{j^*} - T_2[j^*])$ , party  $P_1$  would learn whether or not an error actually occurred, leaking sensitive information. This means that inherently,  $P_2$  must also modify its share in position  $j^*$  as part of the correction procedure. But, this must be done in a way that both hides the identity of  $j^*$ , and also does *not* affect the secret sharing across the two parties in other positions.

This will be done in two pieces: (1)  $P_1$  will learn  $(T_2[j^*] - r)$ , for some secret mask  $r$  chosen by  $P_2$ ; and (2) they will both perform some operation on their local  $T_i$  array that offsets the value shared in position  $j^*$  by exactly  $(p_{j^*} - T_2[j^*])$  while preserving the values shared in all other  $j' \neq j^*$ .

The first of these tasks can be performed by executing a standard single-server polylogarithmic symmetric PIR protocol, where  $P_1$  acts as client with query index  $j^*$ , and  $P_2$  acts as server with the  $r$ -shifted database  $T'_2[j] = T_2[j] - r$ , for random  $r$  of its choice.

The second task will be performed by a low-communication private increment procedure using *distributed point functions* (DPF): namely, FSS for the class of point functions (equivalently, compressed secret shares of a secret unit vector). Actually, since party  $P_1$  knows the identity of  $j^*$ , a weaker tool of punctured PRFs suffice; however, we continue with DPF terminology for notational convenience (both are implied by one-way functions). More concretely, the parties will run a small-scale secure computation protocol on inputs  $j^*, (T_2[j^*] - r)$  (held by  $P_1$ ), the additive shares of  $p_{j^*}$ , and  $r$  (held by  $P_2$ ), which outputs short DPF key shares  $k_1, k_2$  to the respective parties, with the property that  $\text{DPF.Eval}(1, k_1, j) + \text{DPF.Eval}(2, k_2, j) = 0$  for every  $j \neq j^*$ , and  $= (p_{j^*} - T_2[j^*])$  for  $j = j^*$ . Each party thus modifies its  $T_i$  array by offsetting each position  $j$  with the  $j$ th DPF evaluation, yielding precisely the required effect.

This procedure is performed for every flag position  $j^*$ , and for each party  $P_1, P_2$ . (Note that the parties should always perform the above steps  $\lambda$  times, sometimes on dummy values, in order to hide the true number of flagged positions.) The final resulting scheme provides standard FSS correctness guarantees, *removing* the inverse-polynomial error, and thus can be plugged into the approach from above. As mentioned, the new resulting  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  functionality is now a complex procedure, with runtime scaling as the entire truth table size of the shared function. But, the above-described protocol provides a means for securely emulating  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  with *low communication*: scaling just as  $\lambda$ -many small-scale secure computations and PIR executions.

## 2 Preliminaries

### 2.1 Assumptions

We assume familiarity with the following computational assumptions, and refer to the full version for more details: Quadratic Residuosity (QR) [30], Learning With Errors (LWE) [39], Learning Parity with Noise (LPN) [7], Decisional Diffie-Hellman (DDH), and Decision Composite Residuosity (DCR) [38].

### 2.2 Function Secret Sharing and Homomorphic Secret Sharing

We follow the function secret sharing definition of [12], for the specific leakage function which reveals the input and output domain sizes  $(1^n, 1^m)$  of the secret function.

**Definition 6 (Function Secret Sharing (FSS)).** An  $N$ -party Function Secret-Sharing (FSS) scheme (with additive reconstruction) for a function family  $\mathcal{F}$  is a pair of algorithms  $\text{FSS} = (\text{FSS.Gen}, \text{FSS.Eval})$  with the following syntax and properties:

- $\text{Gen}(1^\lambda, \tilde{f})$  is a probabilistic polynomial-time key generation algorithm, which on input  $1^\lambda$  (a security parameter) and  $\tilde{f} \in \{0, 1\}^*$  (the description of some function  $f: \{0, 1\}^n \rightarrow \{0, 1\}^m \in \mathcal{F}$ ), outputs an  $N$ -tuple of keys  $(k_1, \dots, k_N)$ . Each key is assumed to contain  $1^n$  and  $1^m$ .
- $\text{Eval}(i, k_i, x)$  is a deterministic polynomial-time evaluation algorithm, which on input  $i \in [N]$  (the party index),  $k_i$  (a key defining  $f_i: \{0, 1\}^n \rightarrow \{0, 1\}^m$ ), and  $x \in \{0, 1\}^n$  (an input for  $f_i$ ), outputs a value  $y_i \in \{0, 1\}^m$  (the value of  $f_i(x)$ , the  $i^{\text{th}}$  share of  $f(x)$ ).
- **Correctness:** For all  $\lambda \in \mathbb{N}$ , all  $f \in \mathcal{F}$  (described by  $\tilde{f}$ ), and all  $x \in \{0, 1\}^n$ ,

$$\Pr \left[ y_1 + \dots + y_N = f(x) : \begin{array}{l} (k_1, \dots, k_N) \xleftarrow{\$} \text{FSS.Gen}(1^\lambda, \tilde{f}) \\ y_i \leftarrow \text{FSS.Eval}(i, k_i, x), i = 1 \dots N \end{array} \right] = 1 .$$

- **Security:** For every set of corrupted parties  $\mathcal{D} \subsetneq [N]$ , there exists a probabilistic polynomial-time algorithm  $\text{Sim}^{\text{FSS}}$  (a simulator), such that for every sequence of functions  $f_1, f_2, \dots \in \mathcal{F}$  (described by  $\tilde{f}_1, \tilde{f}_2, \dots$ ), the outputs of the following experiments  $\text{Real}^{\text{FSS}}$  and  $\text{Ideal}^{\text{FSS}}$  are computationally indistinguishable:
  - $\text{Real}^{\text{FSS}}(1^\lambda) : (k_1, \dots, k_N) \xleftarrow{\$} \text{Gen}(1^\lambda, \tilde{f}_\lambda)$ ; Output  $(k_i)_{i \in \mathcal{D}}$ .
  - $\text{Ideal}^{\text{FSS}}(1^\lambda) : \text{Output } \text{Sim}^{\text{FSS}}(1^\lambda, 1^N, 1^n, 1^m)$ .

We consider also the dual notion of *Homomorphic Secret Sharing* [11], in which the roles of input and function are reversed, as well as a weaker variant with only Las Vegas inverse-polynomial correctness error.

### 3 General Template for $(N + 1)$ -Party Sublinear Secure Computation from $N$ -Party FSS

In this section we present a generic template for building  $(N + 1)$ -party sublinear secure computation from an  $N$ -party additive function secret sharing scheme (for a well-chosen function class) with two specific properties. We require of the FSS scheme that there exist low-communication protocols to realise the following tasks:

- *$N$ -Party Share Distribution:*  $N$  servers generate FSS shares of some function of their inputs; the ideal functionality  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  is provided in Fig. 2.
- *Two-Party Oblivious Share Evaluation:* A client obliviously evaluates an FSS share held by a server; the ideal functionality  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$  is provided in Fig. 3.

Theorem 7 proves that the protocol provided in Fig. 5 is an  $(N + 1)$ -party secure computation scheme in the  $(\mathcal{F}_{\text{SD}}^{\text{FSS}}, \mathcal{F}_{\text{OE}}^{\text{FSS}})$ -hybrid model. This template achieves sublinear secure computation provided  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  and  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$  can be realised with low enough communication. A high level overview of the protocol is provided in Fig. 1.

### 3.1 Requirements of the FSS Scheme

We start by isolating in the properties we require of the FSS scheme to fit our template for sublinear secure computation, and show that they are satisfied by any additive FSS scheme. At a high level, we require that given a strict subset of the FSS keys, together with the evaluated output shares of all keys on some known input  $x$ , it should be computationally hard to recover any information about the secret shared function  $f$  beyond its evaluation  $f(x)$ . For the formalisation of this property, as well as the proof that additivity suffices, we refer to the full version of the paper.

### 3.2 The Secure Computation Protocol

We define the ideal functionalities  $\mathcal{F}_{SD}^{FSS}$  (Fig. 2) for  $N$ -party FSS share distribution, and  $\mathcal{F}_{OE}^{FSS}$  (Fig. 3) for 2-party oblivious evaluation of FSS shares. We then introduce in Fig. 5 the generic template for secure computation from additive FSS in the  $(\mathcal{F}_{SD}^{FSS}, \mathcal{F}_{OE}^{FSS})$ -hybrid model.

**Functionality** FSS Share Distribution  $\mathcal{F}_{SD}^{FSS}$

**Parameters:** The ideal functionality  $\mathcal{F}_{SD}^{FSS}$  is parameterised by a number of parties  $N$ , a function class  $\mathcal{C} = \{f_{\alpha_1, \dots, \alpha_N}\}_{(\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N}}$ , and an additive FSS scheme  $FSS = (FSS.Gen, FSS.Eval)$  for  $\mathcal{C}$ .

$\mathcal{F}_{SD}^{FSS}$  interacts with the  $N$  parties  $P_1, \dots, P_N$  in the following manner.

**Input:** Wait to receive  $(input, i, x_i)$  where  $x_i \in \{0, 1\}^{\ell_i}$  from each party  $P_i$  (for  $1 \leq i \leq N$ ).

**Output:** Run  $(k_1, \dots, k_N) \stackrel{s}{\leftarrow} FSS.Gen(1^\lambda, \tilde{f}_{x_1, \dots, x_N})$ , where  $\tilde{f}_{x_1, \dots, x_N}$  is a description of  $f_{x_1, \dots, x_N}$ ; Output  $k_i$  to each party  $P_i$  (for  $1 \leq i \leq N$ ).

**Fig. 2.** Ideal functionality  $\mathcal{F}_{SD}^{FSS}$  for the generation of FSS keys of a distributed function.

**Theorem 7 (Template for  $(N+1)$ -Party Sublinear MPC from  $N$ -Party FSS).** *Let  $N \geq 2$ . Let  $C: \mathbb{F}^n \rightarrow \mathbb{F}^m$  be an arithmetic circuit with  $n = \ell_0 + \ell_1 + \dots + \ell_N$  inputs over a finite field  $\mathbb{F}$ , and let  $FSS = (FSS.Gen, FSS.Eval)$  be an (additive) FSS scheme for the following function family of “partial evaluations of  $C$ ”:*

$$\left\{ \begin{array}{l} g_{\alpha_1, \dots, \alpha_N}: \mathbb{F}^{\ell_0} \rightarrow \mathbb{F}^m \\ x \mapsto C(x, \alpha_1, \dots, \alpha_N) \end{array} : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \right\} .$$

**Functionality** Oblivious Evaluation of FSS Shares  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$

**Parameters:** The ideal functionality  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  is parameterised by a number of parties  $N$ , and an additive FSS scheme  $\text{FSS} = (\text{FSS.Gen}, \text{FSS.Eval})$  for some function class  $\mathcal{C}$ .

$\mathcal{F}_{\text{OE}}^{\text{FSS}}$  interacts with two parties, Alice (“the client”) and Bob (“the server”), in the following manner.

**Input:** Wait to receive  $(\text{Client}, x)$  from Alice and  $(\text{Server}, i, k_i)$  from Bob, and record  $(i, k_i, x)$ .

**Output:** Run  $y_i \leftarrow \text{FSS.Eval}(i, k_i, x)$ ; Output  $y_i$  to Alice.

**Fig. 3.** Ideal functionality  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$  for the two-party oblivious evaluation of FSS shares.

**Functionality**  $\mathcal{F}_{\text{SFE}}(C)$

The functionality is parameterised with a number  $N$  and an arithmetic circuit  $C$  with  $n = \ell_0 + \ell_1 + \dots + \ell_N$  inputs and  $m$  outputs over a finite field  $\mathbb{F}$ .

**Input:** Wait to receive  $(\text{input}, i, x_i)$  from each party  $P_i$  ( $0 \leq i \leq N$ ), where  $x_i \in \mathbb{F}^{\ell_i}$ , and set  $\mathbf{x} \leftarrow x_0 \| x_1 \| \dots \| x_N$ .

**Output:** Compute  $\mathbf{y} \leftarrow C(\mathbf{x})$ ; Output  $\mathbf{y}$  to all parties  $P_0, P_1, \dots, P_N$ .

**Fig. 4.** Ideal functionality  $\mathcal{F}_{\text{SFE}}(C)$  for securely evaluating an arithmetic circuit  $C$  among  $N + 1$  parties.

The protocol  $\Pi_C$  provided in Fig. 5 UC-securely implements the  $(N + 1)$ -party functionality  $\mathcal{F}_{\text{SFE}}(C)$  in the  $(\mathcal{F}_{\text{SD}}^{\text{FSS}}(C), \mathcal{F}_{\text{OE}}^{\text{FSS}}(C))$ -hybrid model, against a static passive adversary corrupting at most  $N$  out of  $(N + 1)$  parties. The protocol uses  $N \cdot m \cdot \log |\mathbb{F}|$  bits of communication, and additionally makes one call to  $\mathcal{F}_{\text{SD}}^{\text{FSS}}(C)$  and  $N$  calls to  $\mathcal{F}_{\text{OE}}^{\text{FSS}}(C)$ .

We refer the reader to the full version of the paper for the proof of Theorem 7.

**Protocol  $\Pi_C$** **Parties:**  $P_0, P_1, \dots, P_N$ **Parameters:** The protocol is parameterised with a number of parties ( $N + 1$ ), an arithmetic circuit  $C: \mathbb{F}^n \rightarrow \mathbb{F}^m$  with  $n = \ell_0 + \ell_1 + \dots + \ell_N$ , and an additive FSS scheme  $\text{FSS} = (\text{FSS.Gen}, \text{FSS.Eval})$  for the following function family of “partial evaluations of  $C$ ”:

$$\left\{ \begin{array}{l} g_{\alpha_1, \dots, \alpha_N}: \mathbb{F}^{\ell_0} \rightarrow \mathbb{F}^m \\ x \mapsto C(x, \alpha_1, \dots, \alpha_N) \end{array} : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \right\} .$$

 $(\text{sid}_1, \dots, \text{sid}_N)$  are  $N$  distinct session ids.**Hybrid Model:** The protocol is defined in the  $(\mathcal{F}_{\text{SD}}^{\text{FSS}}, \mathcal{F}_{\text{OE}}^{\text{FSS}})$ -hybrid model.**Input:** Each party  $P_i$  holds input  $x_i \in \mathbb{F}^{\ell_i}$ .**The Protocol:**

1. Each party  $P_i$  for  $i \neq 0$  sends  $(\text{input}, i, x_i)$  to  $\mathcal{F}_{\text{SD}}^{\text{FSS}}(C)$ , and waits to receive  $k_i$ .
2. For each  $i = 1, \dots, N$ :
  - (a) Party  $P_0$  sends  $(\text{sid}_i, \text{Client}, x_0)$  to  $\mathcal{F}_{\text{OE}}^{\text{FSS}}(C)$  and  $P_i$  sends  $(\text{sid}_i, \text{Server}, i, k_i)$  to  $\mathcal{F}_{\text{OE}}^{\text{FSS}}(C)$
  - (b) Party  $P_0$  waits to receive  $(\text{sid}_i, y_i)$  from  $\mathcal{F}_{\text{OE}}^{\text{FSS}}(C)$ .
3. Party  $P_0$  sets  $\mathbf{y} \leftarrow y_1 + \dots + y_N$ , and sends  $\mathbf{y}$  to all parties.
4. Every party outputs  $\mathbf{y}$ .

**Fig. 5.** (Sublinear) secure computation protocol in the  $(\mathcal{F}_{\text{SD}}^{\text{FSS}}, \mathcal{F}_{\text{OE}}^{\text{FSS}})$ -hybrid.

## 4 Oblivious Evaluation of LogLog-Depth FSS from PIR

In the previous section we provided a generic template for  $(N+1)$ -party sublinear secure computation from  $N$ -party additive function secret sharing for which  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  and  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$  can be securely realised with low communication. In this section we introduce the notion of *loglog-depth* for (additive) FSS schemes, and show that this property allows  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$  to be securely realised with low communication using *correlated symmetric PIR* (*corrSPIR*), a primitive introduced in [9] (and which can be instantiated from standard assumptions using the rate-1 batch OT from [17]).

### 4.1 LogLog-Depth FSS

A depth- $d$ ,  $n$ -input,  $m$ -output arithmetic circuit with gates of fan-in at most two over a finite field  $\mathbb{F}$  can be associated with the degree- $(\leq 2^d)$   $n$ -variate  $m$ -output<sup>5</sup> polynomial with coefficients in  $\mathbb{F}$  that it computes. In all generality, a degree- $2^d$   $n$ -variate polynomial can have up to  $\text{nb}_{n,2^d} = \sum_{k=0}^{2^d} \binom{k+n-1}{n-1}$  different monomials (which can be verified using a stars-and-bars counting argument). In this section we will only be interested in circuits whose representation as a polynomial is the sum of  $\text{poly}(\lambda)$  monomials (where  $\lambda$  is the security parameter). A sufficient condition is for it to have  $n = \text{poly}(\lambda)$  inputs and depth  $d \leq \log \log(n)$ ; we refer to this property as a circuit being “of loglog-depth”. Indeed, because we only consider circuits whose gates have fan-in at most two, if a circuit has depth  $d$  then it is  $2^d$ -local (*i.e.* each of its  $m$  outputs is a function of only at most  $2^d$  inputs). Therefore each of its outputs is computed by a polynomial with at most  $\text{nb}_{2^d,2^d} \leq 2^{2^d+2^d}$  monomials, which is  $\text{poly}(\lambda)$  if  $d = \log \log n = \log \log \lambda + \mathcal{O}(1)$ .

We extend in Definition 8 the above notion of “loglog-depth” circuits to “loglog-depth” FSS schemes.

**Definition 8 (LogLog-Depth FSS).** *Let  $\mathcal{F}$  be a class of functions with  $n$  inputs and  $m$  outputs over a finite field  $\mathbb{F}$ . We say that an  $N$ -party FSS scheme  $\text{FSS} = (\text{FSS.Gen}, \text{FSS.Eval})$  for  $\mathcal{F}$  whose evaluation algorithm  $\text{FSS.Eval}$  is explicitly described as an arithmetic circuit, has loglog-depth (alternatively, FSS is a loglog-depth function secret sharing scheme) if for every party index  $i \in [N]$  and every key  $k_i \in \text{Supp}([\text{FSS.Gen}]_i)$  the circuit  $\text{FSS.Eval}(i, k_i, \cdot)$  (which has hardcoded  $i$  and  $k_i$ ) has depth  $\log \log(n)$ .*

Throughout this section we will be using “loglog-depth” circuits and FSS schemes, but it should be noted that all of our results go through if this is replaced everywhere with the more obtuse notion of “circuits (*resp.* FSS evaluation) whose polynomial representation has a polynomial number of coefficients”.

When considering “loglog-depth”, which in particular are “log-local” circuits, we will be interested in the log-sized subsets of the inputs on which each output depends. We say that an FSS scheme is  $(S_1, \dots, S_m)$ -local if the  $j^{\text{th}}$  output of  $\text{FSS.Eval}$ , which takes as input a party index  $i$ , a key  $k_i$ , and an input  $x$ , only depends on  $(i, k_i, x[S_j])$ . In other words, an FSS scheme is  $(S_1, \dots, S_m)$ -local if its evaluation algorithm is  $(S_1, \dots, S_m)$ -local in its last input. We refer to the full version of this paper for a more complete treatment. We emphasize that a loglog-depth circuit or FSS scheme is always log-local, but that the converse is not necessarily true if  $\mathbb{F} \neq \mathbb{F}_2$ .

### 4.2 Oblivious Evaluation of LogLog-Depth FSS from PIR

We first discuss the notion of PIR we need, then show how it can be leveraged to build oblivious evaluation of any loglog-depth FSS scheme.

<sup>5</sup> An  $m$ -output (multivariate) polynomial can be seen as a tuple of  $m$  (multivariate) polynomials.

### 4.2.1 Correlated PIR

We refer to [9] for the definition of the ideal functionality for *batch SPIR with correlated “mix and match” queries* ( $\mathcal{F}_{\text{corrSPIR}}$ ), which we extend in the full version of this paper from the boolean to the arithmetic setting as *batch Oblivious (Multivariate) Polynomial Evaluation with correlated “mix and match” queries* ( $\mathcal{F}_{\text{corrOPE}}$ ).

In the boolean world, this corresponds to a batched form of SPIR, querying into  $k$  size- $N$  databases, where the queries are not independent. Rather, the queried indices can be reconstructed via a public function that “mixes and matches” individual bits of a single bitstring  $\alpha = (\alpha_1, \dots, \alpha_w)$  of length  $w < k \log N$ , in a public manner. What this means is that each of the  $(n = \log N)$ -bit queries to a single database can be obtained by concatenating  $n$  of the bits  $\alpha_i$ , possibly permuted. In the arithmetic world, this corresponds to batch multivariate OPE, where each database corresponds to a polynomial, and the evaluation inputs are various subvectors of some *joint input vector*, comprised of  $w$  field elements. More specifically, the input to a single  $d$ -variate polynomial (in the batch to be obliviously evaluated) is a size- $d$  ordered subset of the joint inputs.

We will be interested in how many times a given bit of entropy (*resp.* input)  $\alpha_i$  appears within the  $k$  queries (*resp.* input)—counted by the occurrence function  $t_i$  below—, as well as how many times it appears in specific index position  $j' \in [n]$  within the  $k$  queries (*resp.* input)—denoted below by  $t_{i,j'}$ —. To the best of our knowledge, there are no protocols realising *corrOPE* over superpolynomial-size fields without FHE, and the only protocol realising *corrSPIR* without FHE requires introducing this notion of “balance between the queried bits”.

We refer to [9] for the formalisation of “mix and match” functions, (the ideal functionality for) batch SPIR with correlated “mix and match” queries ( $\mathcal{F}_{\text{corrSPIR}}$ ), and to the full version of this paper for the (ideal functionality for) batch Oblivious (Multivariate) Polynomial Evaluation with correlated “mix and match” queries ( $\mathcal{F}_{\text{corrOPE}}$ ).

### 4.2.2 Oblivious Evaluation of LogLog-Depth FSS from PIR

Let  $\text{FSS} = (\text{FSS.Gen}, \text{FSS.Eval})$  be a loglog-depth,  $(S_1, \dots, S_m)$ -local FSS scheme (Definition 8). Because FSS has loglog-depth, the polynomial representation of  $\text{FSS.Eval}$  has  $m \cdot \text{poly}(n)$  coefficients. Furthermore, each of its local evaluation algorithms  $\text{FSS.Eval}_j$  depends only on the inputs indexed by  $S_j$ . Therefore obliviously evaluating  $\text{FSS.Eval}$  can be done by using batch OPE with correlated “mix and match” inputs: the  $m$  polynomials in the batch are the  $\text{FSS.Eval}_j(i, k_i, \cdot)$ , where  $k_i$  is known only to the server  $P_i$ . This protocol is formalised in Fig. 6.

Note that this notion of *corrOPE*, as defined in the full version of the paper, requires the polynomials in the batch be represented as a vector of coefficients. For this reason we impose that FSS be loglog-depth, so this vector be polynomial-size.



**Protocol** Oblivious Evaluation of Partial Function Shares  $\Pi_{OE}$

**Parties:**  $P_0$  (the client) and  $P_i$  (the server).

**Parameters:**

- Let  $N$  be a number, and let  $C = C_1 \parallel \dots \parallel C_m$  be a loglog-depth circuit (definition 8) with  $n = \ell_0 + \ell_1 + \dots + \ell_N$  inputs and  $m$  outputs over  $\mathbb{F}$  such that the following function family  $\mathcal{C}$  is  $(S_1, \dots, S_m)$ -local, where  $S_1, \dots, S_m$  is some family of  $(\log / \log \log)$ -sized subsets of  $[n]$ :

$$\mathcal{C} = \left\{ \begin{array}{l} g_{\alpha_1, \dots, \alpha_N} : \mathbb{F}^{\ell_0} \rightarrow \mathbb{F}^m \\ x \mapsto C(x, \alpha_1, \dots, \alpha_N) : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \end{array} \right\}.$$

We assume that each of the  $S_i$  is ordered in such a way that the function `MixAndMatch` associated with  $(S_1, \dots, S_m)$  is polylog-balanced<sup>a</sup> (c.f. full version).

- $\text{FSS} = (\text{FSS.Gen}, \text{FSS.Eval})$  is an  $(S_1, \dots, S_m)$ -local (additive) FSS scheme for  $\mathcal{C}$ , whose local evaluation algorithms (c.f. full version) are  $(\text{FSS.Eval}_j)_{j \in [m]}$ .

**Hybrid Model:** The protocol is defined in the  $\mathcal{F}_{\text{corrOPE}}$ -hybrid model (the subsets characterising `MixAndMatch`, and in turn `corrOPE`, are  $(S_1, \dots, S_m)$ ).

**Input:**  $P_0$  holds input  $x_0 \in \{0, 1\}^{\ell_0}$ , and  $P_i$  holds  $k_i$ .

**The Protocol:**

- **First Round:**

1.  $P_0$  sends **(receiver,  $x_0$ )** to  $\mathcal{F}_{\text{corrOPE}}$
2.  $P_i$  sends **(sender,  $(c_j)_{j \in [m]}$ )** to  $\mathcal{F}_{\text{corrOPE}}$  where  $c_j$  is the vector of coefficients of  $\text{FSS.Eval}_j(i, k_i, \cdot)$   
 // For the case  $\mathbb{F} = \mathbb{F}_2$  (i.e. when using  $\mathcal{F}_{\text{corrSPIR}}$ ), the databases can be more simply described as the truth tables of the  $\text{FSS.Eval}_j(i, k_i, \cdot)$  for  $j \in [m]$ , i.e.  $(\text{FSS.Eval}_j(i, k_i, x'))_{x' \in \{0, 1\}^{|S_j|}}$ .

3. **Second Round:**

4.  $P_0$  waits to receive  $(y_{i,1}, \dots, y_{i,m})$  from  $\mathcal{F}_{\text{corrOPE}}$
5.  $P_0$  outputs  $(y_{i,1}, \dots, y_{i,m})$

<sup>a</sup> By [9, Lemma 9], such orderings exist and furthermore can be found in expected constant time by random shuffling. Alternatively, since a random ordering of  $(S_1, \dots, S_m)$  works with high probability, the protocol could be modified so that  $P_0$  samples a PRG key and sends it to  $P_1$ , and both use the resulting pseudorandomness to order  $(S_1, \dots, S_m)$ . This additional step incurs only a small additive overhead in communication, and the resulting protocol is still sublinear.

**Fig. 6.** Two-party protocol for obviously evaluating shares of an additive loglog-depth FSS scheme.

**Lemma 9 (Oblivious Share Evaluation for LogLog-Depth FSS Schemes).** *Let  $N \geq 2$ . Let  $C: \mathbb{F}^n \rightarrow \mathbb{F}^m$  be a loglog-depth arithmetic circuit with  $n = \ell_0 + \ell_1 + \dots + \ell_N$  inputs over a finite field  $\mathbb{F}$ , and let  $\mathcal{C}$  be the family of “partial evaluations of  $C$ ”:*

$$\left\{ \begin{array}{l} g_{\alpha_1, \dots, \alpha_N}: \mathbb{F}^{\ell_0} \rightarrow \mathbb{F}^m \\ x \mapsto C(x, \alpha_1, \dots, \alpha_N) : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \end{array} \right\} .$$

*If FSS is an additive loglog-depth,  $(S_1, \dots, S_m)$ -local FSS scheme (Definition 8) for  $\mathcal{C}$  and corrSPIR is a two-round batch SPIR protocol (characterised by  $(S_1, \dots, S_m)$ ), then the protocol  $\Pi_{OE}$  provided in Fig. 6 UC-securely implements the two-party functionality  $\mathcal{F}_{OE}^{FSS}$  against a static, passive adversary in the  $\mathcal{F}_{\text{corrOPE}}$ -hybrid model.*

The proof of Lemma 9 is given in the full version of the paper.

## 5 LogLog-Depth FSS from Compact and Additive HSS

In this section we show how to use compact and additive HSS to build a loglog-depth FSS scheme whose share distribution  $\mathcal{F}_{SD}^{FSS}$  can be realised in low communication. When combined with Sects. 3 to 4, this yields sublinear secure computation from compact and additive HSS. In the full version of this paper, we show how to extend this construction to use the weaker primitive of Las-Vegas HSS. We note that this extension forms a non-trivial technical contribution of our work, which we defer to the full version due to lack of space, and focus this edition of the paper on the simplest version of our template.

### 5.1 An Overview of the Construction

Let  $C: \mathbb{F}^n \rightarrow \mathbb{F}^m$  be a log log-depth arithmetic circuit with  $n = \ell_0 + \ell_1 + \dots + \ell_N$  inputs over a finite field  $\mathbb{F}$ , and let  $\mathcal{C}$  be the family of “partial evaluations of  $C$ ”:

$$\left\{ \begin{array}{l} g_{\alpha_1, \dots, \alpha_N}: \mathbb{F}^{\ell_0} \rightarrow \mathbb{F}^m \\ x \mapsto C(x, \alpha_1, \dots, \alpha_N) : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \end{array} \right\} .$$

Our goal in this section is to provide a construction of a loglog-depth FSS scheme for  $\mathcal{C}$  such that  $\mathcal{F}_{SD}^{FSS}$  can be realised with low communication, and we do so by using compact and additive single-function HSS for any function in a well-chosen function class (that of  $\{\text{coefs}_{c_1, \dots, c_N}: (c_1, \dots, c_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N}\}$ , as defined below).

We provide in Fig. 7 a construction of loglog-depth additive FSS for  $\mathcal{C}$  from single-function additive HSS for the following function coefs:

$$\begin{aligned} \text{coefs}: \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} &\rightarrow \mathbb{F}^* \\ (\alpha_1, \dots, \alpha_N) &\mapsto (p_0, p_1, \dots) \end{aligned}$$

where  $(p_0, p_1, \dots)$  are the coefficients of the polynomial representation of all the  $C_j(X, \alpha_1, \dots, \alpha_N)$ , for  $j \in [m]$  (which are polynomials in  $X$ , whose coefficients

are themselves polynomials in  $\alpha_1, \dots, \alpha_N$ ). Because  $C$  has loglog-depth (Definition 8), there are at most  $m \cdot n \cdot (1 + o(1))$  such coefficients. Furthermore, the key generation algorithm of the FSS scheme for  $C$  essentially boils down to a single call to the share generation algorithm of the HSS scheme for  $\text{coefs}$ . Therefore, we also need to provide an HSS scheme for  $\text{coefs}$  whose share generation can be distributed using low communication. We use a transformation akin to hybrid encryption in order to ensure this last property: we mask the inputs using pseudorandom generators, and reduce the problem of generating HSS shares of the inputs to that of distributing HSS shares of the keys, which can be done generically using oblivious transfer.

More precisely, for  $i \in [N]$  let  $G_i: \{0, 1\}^\lambda \rightarrow \mathbb{F}^{\ell_i}$  be a PRG and consider the function family  $\{\text{coefs}_{c_1, \dots, c_N} : (c_1, \dots, c_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N}\}$ , where:

$$\begin{aligned} \text{coefs}_{c_1, \dots, c_N} : \mathbb{F}^\lambda \times \dots \times \mathbb{F}^\lambda &\rightarrow \mathbb{F}^* \\ (K_1, \dots, K_N) &\mapsto (p_0, p_1, \dots) \end{aligned}$$

where  $(p_0, p_1, \dots)$  are the coefficients of the polynomial representation of all the  $C_j(X, c_1 - G_1(K_1), \dots, c_N - G_N(K_N))$ ,  $j \in [m]$  (which are polynomials in  $X$ , whose coefficients are polynomials in the bits of  $K_1, \dots, K_N$ ).

We provide in the full version of this paper the construction of an HSS scheme for  $\text{coefs}$  whose share generation can be distributed using low communication (along with a corresponding protocol), assuming the existence of compact and additive single-function HSS for any function in  $\{\text{coefs}_{c_1, \dots, c_N}\}$ .

While this assumption relating to the existence of HSS for  $\{\text{coefs}_{c_1, \dots, c_N}\}$  may not seem standard, it is weaker than each of the following assumptions:

1. HSS for  $\text{NC}^1$  and polynomial-stretch PRGs in  $\text{NC}^1$ ;
2. Single-function HSS for any log log-depth circuit and constant-depth PRGs with some fixed polynomial-stretch.

## 5.2 Defining the LogLog-Depth FSS Scheme

**Observation 1** (Parsing Additive Shares). *Let  $\mathbf{x} \in \{0, 1\}^n$  and let  $I \subseteq [n]$ . If  $(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})$  are additive shares of  $\mathbf{x}$ , then  $([\mathbf{x}^{(1)}]_I, \dots, [\mathbf{x}^{(m)}]_I)$  are additive shares of  $[\mathbf{x}]_I$ , where  $[\cdot]_I$  denotes the subvector induced by the set of coordinates  $I$ .*

### LogLog-Depth FSS Scheme from Additive HSS

**Parameters:** Let  $N \geq 2$  be a number of parties, and let  $C = C_1 \parallel \dots \parallel C_m$  be a loglog-depth circuit with  $n = \ell_0 + \ell_1 + \dots + \ell_N$  inputs and  $m$  outputs over  $\mathbb{F}$  such that the following function family is  $(S_1, \dots, S_m)$ -local, where  $S_1, \dots, S_m$  is some family of  $(\log n / \log \log n)$ -sized subsets of  $[n]$ :

$$\mathcal{C} = \left\{ \begin{array}{l} g_{\alpha_1, \dots, \alpha_N} : \mathbb{F}^{\ell_0} \rightarrow \mathbb{F}^m \\ x \mapsto C(x, \alpha_1, \dots, \alpha_N) : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \end{array} \right\} .$$

Let  $\text{HSS} = (\text{HSS.Share}, \text{HSS.Eval})$  be an  $N$ -party additive (single-function) HSS scheme for the function:

$$\begin{array}{l} \text{coefs: } \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \rightarrow \mathbb{F}^* \\ (\alpha_1, \dots, \alpha_N) \mapsto (p_0, p_1, \dots) \end{array}$$

where  $(p_0, p_1, \dots)$  are the coefficients of the polynomial representation of all the  $C_j(X, \alpha_1, \dots, \alpha_N)$ ,  $j \in [m]$  (which are polynomials in  $X$ , whose coefficients are themselves polynomials in  $\alpha_1, \dots, \alpha_N$ ).

// Note that since  $C$  has loglog-depth and  $\mathcal{C}$  is  $(S_1, \dots, S_m)$ -local, each of the  $m$  polynomials has degree  $|S_j|$  and  $|S_j|$  variables, and there are therefore at most  $\sum_{j=1}^m \binom{|S_j| + |S_j|}{|S_j|} = m \cdot n \cdot (1 + o(1))$  coefficients, regardless of  $(\alpha_1, \dots, \alpha_N)$ .

**FSS.Gen** $(1^\lambda, \tilde{g}_{\alpha_1, \dots, \alpha_N})$ :

1. Parse  $\tilde{g}_{\alpha_1, \dots, \alpha_N}$  to retrieve  $(\alpha_1, \dots, \alpha_N)$
2.  $(k_1, \dots, k_N) \stackrel{\$}{\leftarrow} \text{HSS.Share}(1^\lambda, i, (\alpha_1, \dots, \alpha_N))$
3. Output  $(k_1, \dots, k_N)$

**FSS.Eval<sub>j</sub>** $(i, k_i, x')$ : //  $x' \in \mathbb{F}^{|S_j|}$  should be seen as an  $S_j$ -subset of some larger  $x \in \mathbb{F}^{\ell_0}$  (i.e.  $x' = x[S_j]$ ), input of **FSS.Eval**.

1.  $(p_{0,i}, p_{1,i}, \dots) \stackrel{\$}{\leftarrow} \text{HSS.Eval}(i, k_i)$
2. Parse  $(p_{0,i}, p_{1,i}, \dots)$  to retrieve shares  $(q_{0,i}, q_{1,i}, \dots)$  of the coefficients of  $C_j(\cdot, \alpha_1, \dots, \alpha_N)$  (c.f. observation 1).
3.  $y_{i,j} \leftarrow (x')^{\otimes |S_j|} \cdot (q_{0,i}, q_{1,i}, \dots)^\top$
4. Output  $y_{i,j}$

**FSS.Eval** $(i, k_i, x)$ :

1. For  $j \in [m]$ , set  $y_{i,j} \leftarrow \text{FSS.Eval}_j(i, k_i, x[S_j])$
2. Output  $(y_{i,j})_{j \in [m]}$

**Fig. 7.** LogLog-Depth FSS Scheme from “Single-Function” Additive HSS for every LogLog-Depth Circuit.

**Lemma 10 (LogLog-Depth FSS Scheme from “Single-Function” Additive HSS).** *Let  $N \geq 2$  be a number of parties, and let  $C = C_1 \parallel \dots \parallel C_m$  be a loglog-depth circuit with  $n = \ell_0 + \ell_1 + \dots + \ell_N$  inputs and  $m$  outputs over  $\mathbb{F}$  such that the following function family is  $(S_1, \dots, S_m)$ -local, where  $S_1, \dots, S_m$  is some family of  $(\log n / \log \log n)$ -sized subsets of  $[n]$ :*

$$\mathcal{C} = \left\{ \begin{array}{l} g_{\alpha_1, \dots, \alpha_N} : \mathbb{F}^{\ell_0} \rightarrow \mathbb{F}^m \\ x \mapsto C(x, \alpha_1, \dots, \alpha_N) : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \end{array} \right\} .$$

Let  $\text{HSS} = (\text{HSS.Share}, \text{HSS.Eval})$  be an  $N$ -party (single-function) additive HSS scheme for the function:

$$\begin{array}{ll} \text{coefs: } \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \rightarrow \mathbb{F}^* & \text{where } (p_0, p_1, \dots) \text{ are the} \\ (\alpha_1, \dots, \alpha_N) \mapsto (p_0, p_1, \dots) & \text{coefficients of the polyno-} \\ & \text{mial representation of all the} \\ & C_j(\cdot, \alpha_1, \dots, \alpha_N), j \in [m]. \end{array}$$

Then the construction of Fig. 7 is an  $N$ -party additive loglog-depth and  $(S_1, \dots, S_m)$ -local FSS scheme for  $\mathcal{C}$ .

The proof of Lemma 10 is deferred to the full version of the paper.

### 5.3 Securely Realising $\mathcal{F}_{\text{SD}}^{\text{FSS}}$ in Low Communication

The FSS scheme  $\text{FSS} = (\text{FSS.Gen}, \text{FSS.Eval})$  of Fig. 7 is parameterised by an additive single-function HSS scheme for the function  $\text{coefs}$ . We provide in the full version of the paper such an HSS scheme with the additional property that it yields FSS for which  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  can be securely realised in low communication.

As explained in the overview of Sect. 5.1, we use a standard hybrid encryption trick in order to build HSS for  $\text{coefs}$  from HSS for  $\{\text{coefs}_{c_1, \dots, c_N}\}$ . This allows us to securely distribute the shares of HSS for  $\text{coefs}$  (and hence the keys of the FSS scheme of Fig. 7) by using generic secure computation to distribute the shares of HSS for  $\{\text{coefs}_{c_1, \dots, c_N}\}$  (which can be done in complexity  $\text{poly}(\lambda + N)$ ). We refer to the full version of the paper for the details.

**Lemma 11 ( $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  for the LogLog-Depth FSS scheme of fig. 7 can be realised with low communication).** *Let  $N \geq 2$  be a number of parties, and let  $C = C_1 \parallel \dots \parallel C_m$  be a loglog-depth circuit with  $n = \ell_0 + \ell_1 + \dots + \ell_N$  inputs and  $m$  outputs over  $\mathbb{F}$  such that the following function family is  $(S_1, \dots, S_m)$ -local, where  $S_1, \dots, S_m$  is some family of  $(\log / \log \log)$ -sized subsets of  $[n]$ :*

$$\mathcal{C} = \left\{ \begin{array}{l} g_{\alpha_1, \dots, \alpha_N} : \mathbb{F}^{\ell_0} \rightarrow \mathbb{F}^m \\ x \mapsto C(x, \alpha_1, \dots, \alpha_N) : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \end{array} \right\} .$$

For  $i \in [N]$ , let  $G_i: \{0, 1\}^\lambda \rightarrow \mathbb{F}^{\ell_i}$  be a constant-depth PRG.

Let  $\text{HSS} = (\text{HSS.Share}, \text{HSS.Eval})$  be an  $N$ -party compact and additive HSS scheme for any function in  $\{\text{coefs}_{c_1, \dots, c_N} : (c_1, \dots, c_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N}\}$ , where:

$$\text{coefs}_{c_1, \dots, c_N} : \mathbb{F}^\lambda \times \dots \times \mathbb{F}^\lambda \rightarrow \mathbb{F}^* \quad \text{where } (p_0, p_1, \dots) \text{ are the coefficients of}$$

$$(K_1, \dots, K_N) \mapsto (p_0, p_1, \dots) \quad \text{the polynomial representation of all the}$$

$$C_j(X, c_1 - G_1(K_1), \dots, c_N - G_N(K_N)),$$

$$j \in [m] \text{ (which are polynomials in } X \text{)}.$$

Then the protocol  $\Pi_{\text{SD}}$  provided in the full version of the paper UC-securely implements the  $N$ -party functionality  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  in the  $\mathcal{F}_{\text{SD}}^{\text{HSS}}$ -hybrid model against a static, passive adversary. Furthermore, assuming oblivious transfer, there exists a protocol (in the real world) using  $(N \cdot \lambda)^{O(1)} + N(N-1) \cdot n \cdot \log |\mathbb{F}|$  bits of communication which UC-securely implements the  $N$ -party functionality  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  against a static, passive adversary.

## 6 Instantiations

In Sect. 6.1, we combine the results of Sects. 3 to 5 and achieve sublinear secure computation from generic assumptions (HSS and forms of PIR/OLE). In Sect. 6.2, we build four-party compact and additive HSS for loglog-depth correlations from standard assumptions (DCR and constant-locality PRGs). In Sect. 6.3, we show how to combine all the above (as well as existing constructions of 2-party HSS) in order to build sublinear secure 3- and 5-party computation from standard assumptions not previously known to imply it (in particular, they are not known to imply FHE).

### 6.1 Sublinear-Communication Secure Multiparty Computation from PIR and Additive HSS

Section 4 established that  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$  for local FSS schemes can be based on batch OPE (with correlated inputs) and Sect. 5 builds local FSS schemes (such that  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  can be realised with low communication) from additive HSS (with or without errors). Plugging these two constructions into the template of Sect. 3 yields sublinear secure multiparty computation from batch OPE and additive HSS.

**Theorem 12 (Sublinear-Communication Secure  $(N+1)$ -Party Computation of Shallow Circuits).** *Let  $N \geq 2$  be a number of parties, and let  $C : \mathbb{F}^n \rightarrow \mathbb{F}^m$  be a depth- $d$  ( $d \leq \log \log n - \log \log \log n$ ) arithmetic circuit with  $n = \ell_0 + \ell_1 + \dots + \ell_N$  inputs over  $\mathbb{F}$ . Assuming the existence of:*

- A family of PRGs  $G_i : \{0, 1\}^\lambda \rightarrow \mathbb{F}^{\ell_i}$  for  $i \in [N]$ ,
- An  $N$ -party compact and additive single-function HSS scheme for any function in the class  $\{\text{coefs}_{\alpha_1, \dots, \alpha_N} : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N}\}$ , where  $\text{coefs}_{x_1, \dots, x_N}$  is the function which, on input  $(K_1, \dots, K_N) \in (\{0, 1\}^\lambda)^N$ , computes the (polynomially many) coefficients of the representation of  $C_j(\cdot, \alpha_1 - G_1(K_1), \dots, \alpha_N - G_N(K_N))$  as  $\ell_0$ -variate polynomials for  $j = 1$  to  $m$ ,
- A protocol for UC-securely realising  $\mathcal{F}_{\text{corrOPE}}$  using communication  $\text{Comm}_{\text{corrOPE}}(k, N_{\text{var}}, \text{deg}, w)$ , where  $k$  is the number of OPEs in the batch,  $N_{\text{var}}$  is the number of variables of each polynomial,  $\text{deg}$  is the degree of each polynomial, and  $w$  is the size of the joint input vector,

There exists a protocol using  $(N + \lambda)^{\mathcal{O}(1)} + N \cdot [(N - 1) \cdot n + m] \cdot \log |\mathbb{F}| + N \cdot \text{Comm}_{\text{corrOPE}}(m, 2^d, 2^d, n)$  bits of communication to securely compute  $C$  amongst  $(N + 1)$  parties (that is, to UC-securely realise  $\mathcal{F}_{\text{SFE}}(C)$ ) in the presence of a semi-honest adversary statically corrupting any number of parties.

*Proof.* The proof of Theorem 12 is obtained by combining the results of Sects. 3 to 5. Our starting point is the generic template of Theorem 7 in the  $(\mathcal{F}_{\text{SD}}^{\text{FSS}}, \mathcal{F}_{\text{OE}}^{\text{FSS}})$ -hybrid model, which uses  $N \cdot m \cdot \log |\mathbb{F}|$  bits of communication and makes a single call to  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$ , and  $N$  to  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$ . We use the FSS scheme of lemmas 10 to 11, for which, by Lemma 9, each call to  $\mathcal{F}_{\text{OE}}^{\text{FSS}}$  can be implemented using communication  $\text{Comm}_{\text{corrOPE}}(m, 2^d, 2^d, n)$  and the single call to  $\mathcal{F}_{\text{SD}}^{\text{FSS}}$  can be implemented using communication  $(N \cdot \lambda)^{\mathcal{O}(1)} + N(N - 1) \cdot n \cdot \log |\mathbb{F}|$ .  $\square$

## 6.2 Four-Party Additive HSS from DCR

In this section, we build a 4-party compact homomorphic secret sharing scheme for the class of loglog-depth circuits. Our starting point is the (non compact) 4-party HSS for constant degree polynomials recently described in [19]. At a high level, the scheme works by *nesting* a 2-party HSS scheme inside another 2-party HSS scheme. Concretely, let  $\text{HSS}_{\text{in}}$  and  $\text{HSS}_{\text{out}}$  be two 2-party HSS schemes. Then, the following is a 4-party HSS:

- $\text{HSS.Share}(x) : \text{run } (x^{(0)}, x^{(1)}) \leftarrow \text{HSS.Share}_{\text{in}}(x)$ . For  $b = 0, 1$ , run  $(x^{(b,0)}, x^{(b,1)}) \leftarrow \text{HSS.Share}_{\text{out}}(x^{(b)})$ . Output  $(x^{(0,0)}, x^{(0,1)}, x^{(1,0)}, x^{(1,1)})$ .
- $\text{HSS.Eval}(i, f, x^{(i)}) : \text{parse } i \text{ as } (b, c) \in \{0, 1\}^2$ . Define  $G_{\text{in}}(f) : x^{(b,c)} \rightarrow \text{HSS}_{\text{in}}.\text{Eval}(b, f, x^{(b,c)})$  and run  $y^{(i)} \leftarrow \text{HSS}_{\text{out}}.\text{Eval}(c, G_{\text{in}}(f), x^{(b,c)})$ .

Therefore, to get 4-party HSS for a function class  $\mathcal{F}$ , we need (1) a 2-party  $\text{HSS}_{\text{in}}$  for  $\mathcal{F}$ , and (2) a 2-party  $\text{HSS}_{\text{out}}$  for the class  $\mathcal{F}' = G_{\text{in}}(\mathcal{F})$ . We refer to the full version of the paper for how to obtain these two building blocks, and now state the resulting theorem in Theorem 13.

**Theorem 13 (Four-Party Additive HSS for Constant-Depth Circuits from DCR).** *Assuming the superpolynomial hardness of DCR and the existence of PRGs with constant locality, there exists a four-party HSS scheme for the class of loglog-depth circuits with  $n_{\text{in}}$  inputs; the HSS scheme has share size  $n_{\text{in}}(1 + o(1))$ . Furthermore, there exists a protocol with communication complexity  $n_{\text{in}} \cdot (4 + o(1))$  (for large enough  $n_{\text{in}}$ ) for securely realising the four-party functionality  $\mathcal{F}_{\text{SD}}^{\text{HSS}}$  for the generation of HSS shares of the concatenation of the parties inputs.*

## 6.3 Sublinear-Communication Secure Multiparty Computation from New Assumptions

Combining Sect. 6.1 with instantiations of  $\text{corrSPIR}$  and additive HSS from the literature (and Sect. 6.2) yields sublinear-communication secure 3- and 5-party computation of shallow boolean circuits from a variety of assumptions. Layered boolean circuits are boolean circuits whose gates can be arranged into layers such that any wire connects adjacent layers. It is well-known from previous works [11, 22, 23] that sublinear protocols for low-depth circuits translate to sublinear protocols for general layered circuits: the parties simply cut the layered circuit into low-depth “chunks”, and securely evaluate it chunk-by-chunk. For each chunk, a sublinear secure protocol is invoked to compute the low-depth function which maps shares of the values on the first layer to shares of the values on the first layer of the next chunk.

**Theorem 14 (Secure  $(N + 1)$ -Party Computation with Sublinear Communication from New Assumptions).**

– **3-PC of Shallow Circuits:** Let  $C: \{0, 1\}^n \rightarrow \{0, 1\}^m$  be a size- $s$ , depth- $d$  ( $d \leq \log \log s - \log \log \log s$ ) boolean circuit. Let  $\epsilon \in (0, 1)$ . Assuming the Learning Parity with Noise (LPN) assumption with dimension  $\text{dim} = \text{poly}(\lambda)$ , number of samples  $\text{num} = (n + m)^{1/3} \cdot \lambda^{\mathcal{O}(1)}$ , and noise rate  $\rho = \text{num}^{\epsilon-1}$  (for some constant  $0 < \epsilon < 1$ ) together with any of the following additional computational assumptions:

- Decisional Diffie-Hellman
- Learning with Errors with polynomial-size modulus
- Quadratic Residuosity and Superpolynomial  $\mathbb{F}_2$ -LPN (i.e. assuming the security against time- $\lambda^{2 \log \lambda}$  adversaries of  $\mathbb{F}_2$ -LPN with dimension  $\lambda^{\log \lambda}$ ,  $2\lambda^{\log \lambda}$  samples, and rate  $\lambda/(2\lambda^{\log \lambda})$ ).

There exists a 3-party protocol with communication complexity  $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + 2^{d+2^d} \cdot \text{poly}(\lambda) \cdot \text{polylog}(n) \cdot ((n + m)^{2/3} + (n + m)^{(1+2\epsilon)/3}))$  to securely compute  $C$  (that is, to UC-securely realise  $\mathcal{F}_{\text{SFE}}(C)$ ) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if  $d \leq (\log \log s)/4$  the communication complexity is  $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + \sqrt{s} \cdot \text{poly}(\lambda) \cdot \text{polylog}(n) \cdot ((n + m)^{2/3} + (n + m)^{(1+2\epsilon)/3}))$  (for some arbitrarily small constant  $0 < \delta < 1/2$ ), which is sublinear in the circuit-size, as detailed in Remark 15.

– **3-PC of Layered Boolean Circuits:** Let  $C: \{0, 1\}^n \rightarrow \{0, 1\}^m$  be a size- $s$ , depth- $d$  layered boolean circuit. Let  $\epsilon \in (0, 1)$ . Assuming the Learning Parity with Noise (LPN) assumption with dimension  $\text{dim} = \text{poly}(\lambda)$ , number of samples  $\text{num} = ((s/d)^2/s^\epsilon)^{1/3} \cdot \text{poly}(\lambda)$ , and noise rate  $\rho = \text{num}^{-1/2}$  together with any of the following additional computational assumptions:

- Decisional Diffie-Hellman
- Learning with Errors with polynomial-size modulus
- Quadratic Residuosity and Superpolynomial  $\mathbb{F}_2$ -LPN (i.e. assuming the security against time- $\lambda^{2 \log \lambda}$  adversaries of  $\mathbb{F}_2$ -LPN with dimension  $\lambda^{\log \lambda}$ ,  $2\lambda^{\log \lambda}$  samples, and rate  $\lambda/(2\lambda^{\log \lambda})$ ).

There exists a 3-party protocol with communication complexity  $\mathcal{O}(n + m + d^{1/3} \cdot s^{2(1+\epsilon)/3} \cdot \text{poly}(\lambda) + s/(\log \log s))$  to securely compute  $C$  (that is, to UC-securely realise  $\mathcal{F}_{\text{SFE}}(C)$ ) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if  $d = o(s^{1-\epsilon}/\text{poly}(\lambda))$  (i.e. the circuit is not too “tall and skinny”) the communication complexity is  $\mathcal{O}(n + m + \frac{s}{\log \log s})$ , which is sublinear in the circuit-size.

– **5-PC of Shallow Circuits:** Let  $\epsilon \in (0, 1)$ . Assuming the existence of a constant-locality PRG with polynomial stretch, there exists a constant  $c \geq 3$  such that for any boolean circuit  $C: \{0, 1\}^n \rightarrow \{0, 1\}^m$  of size  $s$  and depth  $d$  ( $d \leq (\log \log s - \log \log \log s)/2^c$ ), assuming the superpolynomial Decision Composite Residuosity (DCR) assumption, the Learning Parity with Noise (LPN) assumption with dimension  $\text{dim} = \text{poly}(\lambda)$ , number of samples  $\text{num} = (n + m)^{1/3} \cdot \lambda^{\mathcal{O}(1)}$ , and noise rate  $\rho = \text{num}^{\epsilon-1}$  (for some constant  $0 < \epsilon < 1$ ), as well as any of the following computational assumptions:

- Decisional Diffie-Hellman (DDH)
- Learning with Errors with polynomial-size modulus (poly-modulus LWE)
- Quadratic Residuosity (QR) and Superpolynomial  $\mathbb{F}_2$ -LPN (i.e. assuming the security against time- $\lambda^{2 \log \lambda}$  adversaries of  $\mathbb{F}_2$ -LPN with dimension  $\lambda^{\log \lambda}$ ,  $2\lambda^{\log \lambda}$  samples, and rate  $\lambda/(2\lambda^{\log \lambda})$ ).



There exists a 5-party protocol with communication complexity  $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + 2^{d/2^c + 2^{d/2^c}} \cdot \text{poly}(\lambda) \cdot \text{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3}))$  to securely compute  $C$  (that is, to UC-securely realise  $\mathcal{F}_{\text{SFE}}(C)$ ) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if  $d \leq (\log \log s)/2^{c+2}$  the communication complexity is  $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + \sqrt{s} \cdot \text{poly}(\lambda) \cdot \text{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3}))$  (for some arbitrarily small constant  $0 < \epsilon < 1/2$ ), which is sublinear in the circuit-size, as detailed in Remark 15.

- **5-PC of Layered Boolean Circuits:** Let  $\epsilon \in (0, 1)$ . Assuming the existence of a constant-locality PRG with polynomial stretch, there exists a constant  $c \geq 3$  such that for any layered boolean circuit  $C: \{0, 1\}^n \rightarrow \{0, 1\}^m$  of size  $s$  and depth  $d$ , assuming the superpolynomial Decision Composite Residuosity (DCR) assumption, assuming the Learning Parity with Noise (LPN) assumption with dimension  $\text{dim} = \text{poly}(\lambda)$ , number of samples  $\text{num} = ((s2^c/d)^2/s^\epsilon)^{1/3} \cdot \text{poly}(\lambda)$ , and noise rate  $\rho = \text{num}^{-1/2}$  together with any of the following additional computational assumptions:

- Decisional Diffie-Hellman
- Learning with Errors with polynomial-size modulus
- Quadratic Residuosity and Superpolynomial  $\mathbb{F}_2$ -LPN (i.e. assuming the security against time- $\lambda^{2 \log \lambda}$  adversaries of  $\mathbb{F}_2$ -LPN with dimension  $\lambda^{\log \lambda}$ ,  $2\lambda^{\log \lambda}$  samples, and rate  $\lambda/(2\lambda^{\log \lambda})$ ).

There exists a 5-party protocol with communication complexity  $\mathcal{O}(n + m + d^{1/3} \cdot s^{2(1+\epsilon)/3} \cdot \text{poly}(\lambda) + s/(\log \log s))$  to securely compute  $C$  (that is, to UC-securely realise  $\mathcal{F}_{\text{SFE}}(C)$ ) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if  $d = o(s^{1-\epsilon}/\text{poly}(\lambda))$  (i.e. the circuit is not too “tall and skinny”) the communication complexity is  $\mathcal{O}(n + m + \frac{s}{\log \log s})$ , which is sublinear in the circuit-size.

Note that combining the works of [17,37] seems to implicitly yield rate-1 batch OT from DCR, and in turn correlated SPIR [9]: if true, the assumptions for sublinear-communication five-party MPC can be simplified to constant-locality PRG, LPN, and superpolynomial DCR (without the need for DDH, LWE, or QR). Since this claim was never made formally, we do not use it.

The proof of Theorem 14 is deferred to the full version of the paper. We conclude by remarking that while this may not be immediately apparent due to the complicated expressions, the above communication complexities do indeed qualify as “sublinear in the circuit-size”.

*Remark 15 (The Expressions of Theorem 14 are Sublinear in the Circuit Size).* Recall that a protocol for securely computing a size- $s$  circuit with  $n$  inputs and  $m$  outputs is sublinear in the circuit-size if its communication complexity is of the form  $\lambda^{\mathcal{O}(1)} + \text{poly}(n + m) + o(s)$ , where  $\text{poly}$  is some fixed polynomial. The communication of our protocols for loglog-depth circuits, both in the 3- and the 5-party case, are sublinear in the circuit-size. For 3PC and 5PC of loglog-depth circuits, the expression is the following:

$$\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + \sqrt{s} \cdot \text{poly}(\lambda) \cdot \text{polylog}(n) \cdot ((n + m)^{2/3} + (n + m)^{(1+2\epsilon)/3})).$$

where  $\epsilon \in (0, 1)$  is some constant tied to the strength of the LPN assumption. Because we view  $s$  as an arbitrarily large polynomial in the security parameter (in other words we are interested in an asymptotic notion of sublinearity), there exists an arbitrarily

small constant  $\delta \in (0, \frac{1}{2})$  such that  $\text{poly}(\lambda) \leq s^\delta$ . Therefore the complexity can be simplified as:

$$\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + s^{\frac{1}{2} + \delta} \cdot \text{polylog}(n) \cdot ((n + m)^{2/3} + (n + m)^{(1+2\epsilon)/3})).$$

Whenever  $s^\delta \geq \text{polylog}(n) \cdot ((n + m)^{2/3} + (n + m)^{(1+2\epsilon)/3})$ , the above expression is  $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + s^{1+2\delta})$ . Whenever  $s^\delta < \text{polylog}(n) \cdot ((n + m)^{2/3} + (n + m)^{(1+2\epsilon)/3})$ , the entire expression is already some fixed polynomial in  $n + m$ . Therefore, our final complexity is of the form  $\lambda^{\mathcal{O}(1)} + \text{poly}_\delta(n + m) + s^{\frac{1}{2} + 2\delta}$ .

**Acknowledgments.** We thank the anonymous reviewers of Eurocrypt 2023 for their helpful comments and careful proofreading, which helped to improve the paper. Elette Boyle and Pierre Meyer were supported by AFOSR Award FA9550-21-1-0046, a Google Research Award, and ERC Project HSS (852952). Geoffroy Couteau was supported by the French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE39-0001 (project SCENE), and by the France 2030 ANR Project ANR22-PECY-003 Secure-Compute.

## References

1. Abram, D., Damgård, I., Orlandi, C., Scholl, P.: An algebraic framework for silent preprocessing with trustless setup and active security. Cryptology ePrint Archive (2022)
2. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-29011-4\\_29](https://doi.org/10.1007/978-3-642-29011-4_29)
3. Barkol, O., Ishai, Y.: Secure computation of constant-depth circuits with applications to database search problems. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 395–411. Springer, Heidelberg (2005). [https://doi.org/10.1007/11535218\\_24](https://doi.org/10.1007/11535218_24)
4. Beaver, D., Feigenbaum, J., Kilian, J., Rogaway, P.: Security with low communication overhead. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 62–76. Springer, Heidelberg (1991). [https://doi.org/10.1007/3-540-38424-3\\_5](https://doi.org/10.1007/3-540-38424-3_5)
5. Belaga, E.G.: Locally synchronous complexity in the light of the trans-box method. In: Fontet, M., Mehlhorn, K. (eds.) STACS 1984. LNCS, vol. 166, pp. 129–139. Springer, Heidelberg (1984). [https://doi.org/10.1007/3-540-12920-0\\_12](https://doi.org/10.1007/3-540-12920-0_12)
6. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC, pp. 1–10. ACM Press (1988). <https://doi.org/10.1145/62212.62213>
7. Blum, A., Furst, M., Kearns, M., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 278–291. Springer, Heidelberg (1994). [https://doi.org/10.1007/3-540-48329-2\\_24](https://doi.org/10.1007/3-540-48329-2_24)
8. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8270, pp. 280–300. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-42045-0\\_15](https://doi.org/10.1007/978-3-642-42045-0_15)

9. Boyle, E., Couteau, G., Meyer, P.: Sublinear secure computation from new assumptions. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022, Part II. LNCS, vol. 13748, pp. 121–150. Springer, Heidelberg (Nov 2022). [https://doi.org/10.1007/978-3-031-22365-5\\_5](https://doi.org/10.1007/978-3-031-22365-5_5)
10. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46803-6\\_12](https://doi.org/10.1007/978-3-662-46803-6_12)
11. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53018-4\\_19](https://doi.org/10.1007/978-3-662-53018-4_19)
12. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: improvements and extensions. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 1292–1303. ACM Press (2016). <https://doi.org/10.1145/2976749.2978429>
13. Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: optimizing rounds, communication, and computation. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 163–193. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-56614-6\\_6](https://doi.org/10.1007/978-3-319-56614-6_6)
14. Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: Karlin, A.R. (ed.) ITCS 2018, vol. 94, pp. 1–21. LIPIcs (2018). <https://doi.org/10.4230/LIPIcs.ITCS.2018.21>
15. 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](https://doi.org/10.1007/978-3-642-54631-0_29)
16. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 3–33. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17656-3\\_1](https://doi.org/10.1007/978-3-030-17656-3_1)
17. Brakerski, Z., Branco, P., Döttling, N., Pu, S.: Batch-OT with optimal rate. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 157–186. Springer, Heidelberg (2022). [https://doi.org/10.1007/978-3-031-07085-3\\_6](https://doi.org/10.1007/978-3-031-07085-3_6)
18. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th ACM STOC, pp. 11–19. ACM Press (1988). <https://doi.org/10.1145/62212.62214>
19. Chillotti, I., Orsini, E., Scholl, P., Smart, N.P., Leeuwen, B.V.: Scooby: improved multi-party homomorphic secret sharing based on FHE. SCN 2022 (2022). <https://eprint.iacr.org/2022/862>
20. Chor, B., Gilboa, N.: Computationally private information retrieval (extended abstract). In: 29th ACM STOC, pp. 304–313. ACM Press (1997). <https://doi.org/10.1145/258533.258609>
21. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: 36th FOCS, pp. 41–50. IEEE Computer Society Press (1995). <https://doi.org/10.1109/SFCS.1995.492461>
22. Couteau, G.: A note on the communication complexity of multiparty computation in the correlated randomness model. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 473–503. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17656-3\\_17](https://doi.org/10.1007/978-3-030-17656-3_17)
23. Couteau, G., Meyer, P.: Breaking the circuit size barrier for secure computation under Quasi-polynomial LPN. In: Canteaut, A., Standaert, F.-X. (eds.) EURO-

- CRYPT 2021. LNCS, vol. 12697, pp. 842–870. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77886-6\\_29](https://doi.org/10.1007/978-3-030-77886-6_29)
24. Damgård, I., Faust, S., Hazay, C.: Secure two-party computation with low communication. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 54–74. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-28914-9\\_4](https://doi.org/10.1007/978-3-642-28914-9_4)
  25. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 93–122. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53015-3\\_4](https://doi.org/10.1007/978-3-662-53015-3_4)
  26. Fazio, N., Gennaro, R., Jafarikhah, T., Skeith, W.E.: Homomorphic secret sharing from Paillier encryption. In: Okamoto, T., Yu, Y., Au, M.H., Li, Y. (eds.) ProvSec 2017. LNCS, vol. 10592, pp. 381–399. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-68637-0\\_23](https://doi.org/10.1007/978-3-319-68637-0_23)
  27. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC, pp. 169–178. ACM Press (2009). <https://doi.org/10.1145/1536414.1536440>
  28. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-642-55220-5\\_35](https://doi.org/10.1007/978-3-642-55220-5_35)
  29. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC, pp. 218–229. ACM Press (1987). <https://doi.org/10.1145/28395.28420>
  30. Goldwasser, S., Micali, S.: Probabilistic encryption and how to play mental poker keeping secret all partial information. In: 14th ACM STOC, pp. 365–377. ACM Press (1982). <https://doi.org/10.1145/800070.802212>
  31. Gál, A., Jang, J.T.: The size and depth of layered boolean circuits. *Inf. Process. Lett.* **111**(5), 213–217 (2011). <https://doi.org/10.1016/j.ipl.2010.11.023>
  32. Harper, L.H.: An log lower bound on synchronous combinational complexity (1977)
  33. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 669–684. ACM Press (2013). <https://doi.org/10.1145/2508859.2516668>
  34. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC, pp. 723–732. ACM Press (1992). <https://doi.org/10.1145/129712.129782>
  35. Kushilevitz, E., Ostrovsky, R.: Replication is NOT needed: SINGLE database, computationally-private information retrieval. In: 38th FOCS, pp. 364–373. IEEE Computer Society Press (1997). <https://doi.org/10.1109/SFCS.1997.646125>
  36. Naor, M., Nissim, K.: Communication preserving protocols for secure function evaluation. In: 33rd ACM STOC, pp. 590–599. ACM Press (2001). <https://doi.org/10.1145/380752.380855>
  37. Orlandi, C., Scholl, P., Yakoubov, S.: The rise of Paillier: homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 678–708. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77870-5\\_24](https://doi.org/10.1007/978-3-030-77870-5_24)
  38. Paillier, P.: Public-key cryptosystems based on composite degree Residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48910-X\\_16](https://doi.org/10.1007/3-540-48910-X_16)
  39. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC, pp. 84–93. ACM Press (2005). <https://doi.org/10.1145/1060590.1060603>

40. Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12827, pp. 687–717. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84252-9\\_23](https://doi.org/10.1007/978-3-030-84252-9_23)
41. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press (1986). <https://doi.org/10.1109/SFCS.1986.25>



# Actively Secure Arithmetic Computation and VOLE with Constant Computational Overhead

Benny Applebaum<sup>(✉)</sup>  and Niv Konstantini

Tel Aviv University, Tel Aviv, Israel  
bennyap@post.tau.ac.il

**Abstract.** We study the complexity of two-party secure arithmetic computation where the goal is to evaluate an arithmetic circuit over a finite field  $\mathbb{F}$  in the presence of an active (aka malicious) adversary. In the passive setting, Applebaum et al. (Crypto 2017) constructed a protocol that only makes a *constant* (amortized) number of field operations per gate. This protocol uses the underlying field  $\mathbb{F}$  as a black box, makes black-box use of (standard) oblivious transfer, and its security is based on arithmetic analogs of well-studied cryptographic assumptions. We present an actively-secure variant of this protocol that achieves, for the first time, all the above features. The protocol relies on the same assumptions and adds only a minor overhead in computation and communication.

Along the way, we construct a highly-efficient Vector Oblivious Linear Evaluation (VOLE) protocol and present several practical and theoretical optimizations, as well as a prototype implementation. Our most efficient variant can achieve an asymptotic rate of  $1/4$  (i.e., for vectors of length  $w$  we send roughly  $4w$  elements of  $\mathbb{F}$ ), which is only slightly worse than the passively-secure protocol whose rate is  $1/3$ . The protocol seems to be practically competitive over fast networks, even for relatively small fields  $\mathbb{F}$  and relatively short vectors. Specifically, our VOLE protocol has 3 rounds, and even for 10K-long vectors, it has an amortized cost per entry of less than 4 OT's and less than 300 arithmetic operations. Most of these operations (about 200) can be pre-processed locally in an offline non-interactive phase. (Better constants can be obtained for longer vectors.) Some of our optimizations rely on a novel intractability assumption regarding the non-malleability of noisy linear codes, that may be of independent interest.

Our technical approach employs two new ingredients. First, we present a new information-theoretic construction of Conditional Disclosure of Secrets (CDS) and show how to use it in order to immunize the VOLE protocol of Applebaum et al. against active adversaries. Second, by using elementary properties of low-degree polynomials, we show that, for some

---

Supported by the Israel Science Foundation grant no. 2805/21. The full version of this paper appears in [10].

simple arithmetic functionalities, one can easily upgrade Yao’s garbled-circuit protocol to the active setting with a minor overhead while preserving the round complexity.

**Keywords:** Foundations · Protocols · Secure Computation

## 1 Introduction

Secure multiparty protocols (MPC) allow a set of parties to jointly compute a function over their inputs while keeping those inputs private. In many situations, the underlying sensitive data is *numerical*, and the computation can be naturally expressed as a sequence of arithmetic operations such as addition, subtraction, and multiplication.<sup>1</sup> This calls for *secure arithmetic computation*, namely secure computation of functions defined by arithmetic operations. It is convenient to represent such a function by an *arithmetic circuit*, which is similar to a standard Boolean circuit except that gates are labeled by addition, subtraction, or multiplication. It is typically sufficient to consider such circuits that evaluate the operations over a large *finite field*  $\mathbb{F}$ , since arithmetic computations over the integers or (bounded precision) reals can be reduced to this case. Computing over finite fields (as opposed to integers or reals) can also be a feature, as it is useful for applications in threshold cryptography (see, e.g., [31]).

It is always possible to reduce arithmetic computation to Boolean computation by implementing each arithmetic operation by Boolean circuit. However, this approach leads to a large blow-up in the circuit size.<sup>2</sup> Thus we strive for “purely arithmetic” solutions that avoid such an emulation. Specifically, following [5], we strive for a protocol that achieves a *constant computational overhead*. That is, we would like to securely evaluate any arithmetic circuit  $C$  over any finite field  $\mathbb{F}$ , with a computational cost (on a RAM machine) that is only a constant times bigger than the cost of performing  $|C|$  field operations with no security at all. Here we make the standard convention of viewing the size of  $C$  also as a security parameter, namely the view of any adversary running in time  $\text{poly}(|C|)$  can be simulated up to a negligible error in  $|C|$ .

### 1.1 ADINZ: Constant Overhead with Passive Security

In [5] (hereafter referred to as ADINZ) it was shown that it is possible to realize 2-party arithmetic MPC with constant computational overhead in the presence of a *passive* (aka semi-honest) adversary. Specifically, ADINZ introduced the *Vector Oblivious Linear Evaluation* (VOLE) functionality in which the sender holds a pair of vectors  $\mathbf{a}, \mathbf{b} \in \mathbb{F}^w$  and the receiver holds a scalar  $x \in \mathbb{F}$  and

<sup>1</sup> More complex numerical computations can typically be efficiently reduced to these simple ones, e.g., by using suitable low-degree approximations.

<sup>2</sup> For example, for the case of finite fields with  $n$ -bit elements, the size of the best known Boolean multiplication circuits is  $\omega(n \log n)$ .

gets as an output the vector  $x\mathbf{a} + \mathbf{b}$ , and showed how to (1) realize VOLE of width  $w$  with complexity of  $O(w)$  and (2) how to use VOLE to realize *batch Oblivious Linear Evaluation* (batch-OLE) of length  $n$  with complexity  $O(n)$ . The latter functionality takes a pair of vectors  $\mathbf{c}, \mathbf{d} \in \mathbb{F}^n$  from the sender and a vector  $\mathbf{y} \in \mathbb{F}^n$  from the receiver and delivers to the receiver the vector  $\mathbf{y} \odot \mathbf{c} + \mathbf{d}$  where  $\odot$  stands for entry-wise multiplication. Both the VOLE functionality and batch-OT functionality naturally extend the Oblivious Linear Evaluation (OLE) functionality [40, 44] that corresponds to the case where  $w = 1$  or  $n = 1$ . Moreover, OLE, VOLE, and batch-OLE can be viewed as the arithmetic versions of oblivious transfer (OT), string-OT, and batch-OT, respectively. Indeed, just like in the binary setting, securely computing an  $\ell$ -size arithmetic circuit reduces via an arithmetic-GMW construction [40] to the task of securely computing batch-OLE of length  $\ell$ . Based on this reduction, ADINZ constructed a constant-overhead MPC protocol for general arithmetic circuits.

The security of the ADINZ protocols is based on arithmetic analogs of well-studied cryptographic assumptions. Concretely, for the VOLE protocol, it suffices to assume the existence of a linear-time computable “code” over  $\mathbb{F}$  for which noisy codewords are pseudorandom. Since a conservative choice of *constant-rate* noise suffices, one can instantiate this LPN-type assumption based on an arithmetic variant of Alekhovich’s assumption [2] or based on the codes of Druk and Ishai [25]. The batch-OLE protocol is based on an arithmetic version of a  $\text{NC}^0$  polynomial-stretch PRG [3, 9, 11, 38]. (See [53] for security analysis of these two assumptions in the arithmetic setting.)

The ADINZ protocols also enjoy several useful properties. They make only black-box access of the field  $\mathbb{F}$  and their arithmetic complexity (the number of field operations) grows linearly with the circuit size of the underlying functionality and is *independent* of the size of  $\mathbb{F}$ .<sup>3</sup> In addition, all protocols make a *black-box* use of a standard OT channel. In fact, in the hybrid-OT model, they achieve information-theoretic privacy against a corrupted VOLE/batch-OLE sender.<sup>4</sup> As advocated in [39, 40], designing protocols in the OT-hybrid model yield several advantages such as native pre-processing [14], simple amortization via OT-extension [12, 14, 34], and the ability to rely on different concrete implementations (including UC-secure ones) under a variety of computational or physical assumptions. Moreover, black-box usage is typically a necessary condition for obtaining practical efficiency. Indeed, the VOLE protocol of ADINZ makes only light-weight linear-algebraic operations and operates in a constant number of rounds, and a prototype implementation appeared in [5]. It should be mentioned that in the past few years the VOLE primitive has turned out to be an important building block with numerous applications such as secure computation of linear algebraic computations [43], round-efficient secure arithmetic computation via arithmetic garbling [8], secure keyword search and set intersection [26, 30], zero-knowledge proofs for arithmetic circuits [13, 17, 19, 23, 51], and non-interactive secure computation [19, 23]. (See [5, 17, 40] and references therein.)

<sup>3</sup> The protocol additionally uses standard “bit-operations” whose complexity is dominated by the field operations.

<sup>4</sup> We mention that the aforementioned assumptions are not known to imply OT.



## 1.2 Actively Secure Arithmetic MPC with Constant Overhead?

Unfortunately, the ADINZ protocols are only passively secure, and, as we will later see, an active adversary can completely break the privacy of both protocols (the VOLE protocol and the batch-OLE). One can probably construct an actively-secure protocol by combining ADINZ with constant-overhead arithmetic zero-knowledge proofs via an arithmetic version of the GMW-compiler [32]. The elegant work of Bootle et al. [15] provides such a zero-knowledge protocol. However, this approach inherently makes a non-BB use of the underlying OT protocol. Also, the protocol of [15] has a super-constant round complexity.

For the special case of VOLE, the breakthrough results of Boyle et al. [17, 18] yield an actively-secure realization with constant overhead. However, their protocols are based on strong LPN-type assumptions with *sub-constant* noise rate. The protocol can be based on OT in a black-box way [18] at the expense of further strengthening the underlying intractability assumption to a *leaky LPN* assumption and by making additional use of *correlation robust hash functions*.

To summarize, to the best of our knowledge, it is currently unknown how to realize batch-OT (or general arithmetic MPC) with active security, constant overhead, and black-box access to OT, regardless of the underlying assumption. For VOLE, the only known constructions either make a non-BB use of OT or rely on relatively strong intractability assumptions such as leaky-LPN with sub-constant noise and correlation robust hash functions. Our goal in this work is to avoid these limitations and derive an actively-secure arithmetic MPC protocol with constant overhead that enjoys all the features of the ADINZ protocol.

## 2 Our Contribution

We resolve the above question in the affirmative by presenting actively-secure variants of the ADINZ protocols for VOLE, batch-OLE, and general arithmetic secure computation, that enjoy all the additional features and are based on the same assumptions. While our main focus is theoretical, we also present several practical optimizations to the VOLE protocol that make use of less conservative intractability assumptions. Details follow.

### 2.1 The VOLE Protocols

Just like [5], we rely on the existence of *fast pseudorandom matrix*  $M \in \mathbb{F}^{m \times k}$  where  $m > k$  is a fixed polynomial in  $k$  (say  $m = k^3$ ). Here “fast” means the mapping  $u \mapsto M \cdot u$  can be computed by making only  $O(m)$  arithmetic operations, and “pseudorandom” means that if we take a random vector in the image of  $M$ , and add a random  $\mu$ -sparse noise vector to it, the resulting vector is computationally indistinguishable from a truly random vector over  $\mathbb{F}^m$ . The noise rate  $\mu$  can be taken to be a constant, e.g.,  $1/4$ . There are several candidates for such fast pseudorandom matrices (see the discussion after Assumption 3). We prove the following theorem.

**Theorem 1 (informal).** *Based on the fast pseudorandom matrix assumption, the VOLE functionality of width  $w$  can be realized with active security in the OT-hybrid model with arithmetic complexity of  $O(w)$  and with perfect security against an active adversary that corrupts the Sender and computational security against the Receiver.*

This protocol (and all the protocols constructed in this work) makes black-box use of the underlying field and is therefore fully arithmetic in the sense of [40]. (One can also derive the stronger form of arithmetic MPC of [4] by instantiating the OT channel with an “arithmetic OT”). In addition, all the protocols that are constructed in this work admit a straight-line black-box simulator. In the 2-party setting, the existence of such simulators implies that the protocol is UC-secure as follows from [41, Theorem 1.5] and [42].

As already mentioned, Theorem 1 is the first to achieve constant overhead and black-box dependency in the OT based on a conservative constant-rate noise LPN-type assumption. Moreover, to the best of our knowledge, this is the first construction that achieves constant overhead and statistical Sender security in the OT-hybrid model, regardless of the underlying assumption,

*Remark 1 (Realizing the OT-Channels with Constant Overhead)* . For the sake of communication/computational complexity, we charge every bit that is passed over the OT channel as a single bit/bit-operation. As already observed in [5, 38], when each OT message is sufficiently long compared to the security parameter (which is the case in our protocols), such OT-channels can be realized securely based on an *arbitrary* OT protocol with the aid of a linear-time computable linear-stretch PRG. The existence of the latter follows from the binary version of the fast pseudorandom matrix assumption (see [5, 7, 38]). In particular, by using any UC-secure OT protocol in the CRS setting (e.g., [46]), we derive UC-secure implementations of our protocols in the standard model. (See the full version for more details.)

*Optimizations: VOLE1 and VOLE2.* Motivated by the rich applications of VOLE, we present several optimizations for the protocol. At the extreme, we present a VOLE protocol (VOLE1) that can achieve an asymptotic rate of  $1/4$  (i.e., the communication is dominated by sending roughly  $4w$  elements of  $\mathbb{F}$ ), which is only slightly worse than the passively-secure protocol whose rate is  $1/3$ . Assuming that the OT consumes 2 rounds, VOLE1 has 3 rounds of computation. The protocol is provably secure against a computationally-unbounded sender, provably secure against a passive receiver, but only heuristically secure against an active receiver. That is, we conjecture that it achieves security against an active receiver, but do not have a security reduction to a clean intractability assumption. As a compromise, we introduce another protocol VOLE2 that slightly downgrades the asymptotic rate of  $1/5$ , has 6 rounds, but can be proved secure based on a new, yet plausible, intractability assumption.<sup>5</sup>

<sup>5</sup> In fact, even our most conservative protocol (VOLE3) that proves Theorem 1 has an asymptotic rate of  $1/5$  and its amortized computational complexity is roughly the same. However, VOLE3 achieves this only over significantly longer vectors.

*The Correlated Noisy-Codeword Hardness Assumption.* Our assumption intuitively asserts that given a random noisy codeword  $\mathbf{c}$  sampled from a code  $T$  with noise pattern  $\mathbf{e}$ , it is hard to efficiently generate a new noisy codeword  $\mathbf{d}$  whose noise pattern  $\mathbf{e}'$  is non-trivially correlated with the noise pattern  $\mathbf{e}$  in the following sense. The new noise vector  $\mathbf{e}'$  is “far” from being a scalar multiple of  $\mathbf{e}$  but it agrees with the original noise vector  $\mathbf{e}$  with respect to the set of non-noisy coordinates  $I = \{i : e_i = 0\}$  of  $\mathbf{e}$ , i.e.,  $\mathbf{d}[I]$  is in the span of  $T[I]$ . Observe that such a noisy codeword can be generated by sampling a vector in the column span of  $(T)\mathbf{c}$  and then modifying  $\ell$  entries with the hope that all these entries fall out of the set of clean coordinates  $I$ . Such an attack succeeds with probability  $\mu^\ell$ , and our assumption states that one cannot do much better than this. (See Sect. 6 for a formal statement.) We believe that this assumption may be of independent interest and provide some evidence towards its validity in the full version [10].

*Implementation and Concrete Complexity.* The computational overhead of VOLE1 and VOLE2 are essentially the same. In the full version [10], we analyze the concrete complexity of these protocols when instantiated with the same building blocks that were used in the passive setting of [5], suggest several practical optimizations, and present an implementation of the protocols. We show that the computational overhead compared to the passive version is minor (less than 20%). Furthermore, even for relatively short vectors of length 10000, our protocols have an amortized cost per VOLE entry of fewer than 4 OTs and less than 300 arithmetic operations (additions and multiplications). We use novel techniques (e.g., sparse LU-decomposition) to push most of these operations (about 200) to a *non-interactive* offline phase that can be pre-processed locally based only on the public parameters and local random tape. As a result, this preprocessing can be applied even before each party knows who will be her partner for the computation. The protocol is also very cheap for the receiver and requires in the online phase less than 10 arithmetic operations and 4 OT’s per VOLE entry. (The sender’s online amortized computation consists of 4 OT’s and less than 80 arithmetic operations per entry.) Such a receiver-efficient protocol is especially useful for applications like VOLE-based zero-knowledge proofs (e.g., [23]) in which the verifier plays the receiver and the prover plays the VOLE sender.

We believe that our protocol may be practically competitive over fast networks even for relatively small fields  $\mathbb{F}$  and relatively short vectors. (Think for example, of an arithmetic zero-knowledge for a circuit that contains few 10K’s gates, which, by [23], translates into a single VOLE of comparable length.) When comparing our protocols to the alternative compressed-VOLE-based solution [17, 18], we see that the latter achieves a better rate of 1/2, but its amortization point seems to “kick in” only for longer vectors (due to the use of an “internal-MPC” protocol for securely realizing “short VOLE-correlations”). Thus, this approach is in a sense complementary to ours, and it will be interesting to

study the combination of the two.<sup>6</sup> We also expect that additional optimizations of our implementation and the underlying building blocks will further improve the computational cost.

## 2.2 The Batch-OLE Protocol

The ADINZ [5] protocol for batch-OLE is based on the existence of *pseudo-random generator* (PRG)  $G : \mathbb{F}^k \rightarrow \mathbb{F}^n$  with polynomial stretch, e.g.,  $n = k^2$ , that is computable by a constant-depth ( $\mathbf{NC}^0$ ) arithmetic circuit.<sup>7</sup> Candidate constructions are studied in [5, 53]. We prove the following theorem.

**Theorem 2 (informal).** *Assuming the existence of an  $\mathbf{NC}^0$  arithmetic PRG with polynomial stretch, the batch-OLE functionality of length  $n$  can be realized with active security in the VOLE-hybrid model with arithmetic complexity of  $O(n)$  and by making a single call to an ideal  $k$ -length  $O(n/k)$ -width VOLE where the batch-OLE receiver (resp., sender) plays the role of the VOLE receiver (resp., sender). The protocol is perfectly secure against an active adversary that corrupts the receiver and computationally secure against an adversary that actively corrupts the sender. Moreover, the protocol has a constant number of rounds.*

Here the  $k$ -length  $O(n/k)$ -width VOLE functionality consists of  $k$  copies of  $O(n/k)$ -width VOLE. The protocol has 4 rounds (counting VOLE as a 2-round protocol). In [27] it is shown that the task of securely computing an arithmetic circuit  $C$  with active security reduces to batch-OLE of length  $O(|C|)$  with constant computational overhead while preserving information-theoretic security. Combining this with Theorems 2 and 1, we derive the following corollary.

**Corollary 1.** *Assuming a fast pseudorandom matrix and an  $\mathbf{NC}^0$  PRG with polynomial stretch, any two-party functionality that is computable by an arithmetic circuit  $C$  can be realized with arithmetic complexity of  $O(|C|)$  in the OT-hybrid model while providing information-theoretic security for one party and computational security for the other party. The protocol makes black-box use of the underlying field.*

The corollary extends to any constant number of parties via standard reductions (e.g., [27]).

---

<sup>6</sup> The current implementations of the compressed-VOLE-based solution are either restricted to the binary field [18] or achieve passive security [48] and so we cannot compare the actual performance of our implementation against a compressed-VOLE-based implementation. Indeed, to the best of our knowledge, it seems that our work provides the first implementation of actively-secure VOLE over large fields.

<sup>7</sup> The exact level of stretch is not important since one can transform a given PRG with a polynomial stretch of  $n = k^c$  for some  $c > 1$ , to a PRG with a stretch of  $n = k^{c'}$  for an arbitrary constant  $c' > c$  while increasing the depth of the circuit by a constant factor (see, e.g., [3]).

### 2.3 Technical Overview of the VOLE Protocols

We briefly present some of the main technical ideas behind our constructions starting with the VOLE protocols.

*The Passive-VOLE Protocol.* The VOLE protocol of ADINZ17 is based on a protocol for “reverse VOLE” (RVOLE) functionality in which Bob holds a vector  $\mathbf{a} \in \mathbb{F}^w$ , Alice holds a vector  $\mathbf{b} \in \mathbb{F}^w$  and a scalar  $x \in \mathbb{F}$  and the goal is to deliver the value of  $x\mathbf{a} + \mathbf{b}$  to Bob. Roughly, Bob sends to Alice an encryption  $\mathbf{c} = E(\mathbf{a}) \in \mathbb{F}^m$  of  $\mathbf{a}$  which is based on the fast pseudorandom matrix. ( $E$  also depends on some random field elements that are omitted for simplicity.) This encryption is “almost-linear” and so Alice can homomorphically compute a new ciphertext  $\mathbf{d} = E(x\mathbf{a} + \mathbf{b}) \in \mathbb{F}^m$  by applying linear operations over  $\mathbf{c}$ . However, this ciphertext cannot be sent to Bob since it leaks information about  $x$  and  $\mathbf{b}$ . In particular, if Bob sees a coordinate of  $\mathbf{d}$  that was “noisy” in the original ciphertext  $\mathbf{c}$  he can efficiently extract the private input of Alice. (See the full version [10].) To fix the problem, we let Bob read only the entries of  $\mathbf{d}$  for which  $\mathbf{c}$  is non-noisy.<sup>8</sup> Of course, Bob has to hide these locations, and so, for each entry  $i \in [m]$ , the parties invoke a standard 1-out-of-2 (or even all-or-nothing [47]) OT-channel where Alice sends the pair  $(\mathbf{d}_i, \perp)$  and Bob’s selection bit determines whether to read  $\mathbf{d}_i$  or to receive a  $\perp$  symbol. While in the passive setting Bob can be trusted to read only the clean locations, an actively corrupt Bob can simply read all the entries of the vector  $\mathbf{d}$ , and completely recover Alice’s input.

*Securing the Protocol Against Active Bob via CDS.* Let us denote by  $T$  the matrix that corresponds to the linear part of the encryption  $E$ , i.e.,  $T$  maps a plaintext of length  $w$  (and a vector of  $k$  random field elements) to a vector of length  $m$  whose noisy version corresponds to a ciphertext. Our first observation is that the above protocol remains secure if and only if the entries  $I \subset [m]$  that Bob reads in the OTs satisfy the following condition: (\*) The ciphertext  $\mathbf{c}$  restricted to  $I$  is in the span of  $T[I]$ , the sub-matrix of  $T$  whose rows are indexed by  $I$ . (As a sanity check, observe that when Bob is honest the set  $I$  of “clean” coordinates satisfies the condition.) At this point, it is natural to try and extend the protocol with some form of zero-knowledge proof in which Bob proves that  $I$  satisfies the above condition. However, since  $I$  corresponds to Bob’s input to the OTs (specifically, Bob’s “selection bits”) such a proof system seems to lead to a non-BB use of the OTs. To avoid this complication, we take an alternative route and make use of a special-tailored Conditional Disclosure of Secret (CDS) Protocol [28].

Roughly speaking, in such a protocol Alice chooses a random secret  $s$  and, for each index  $i \in [m]$ , Alice sends over the  $i$ th OT-call a pair of field elements

<sup>8</sup> This information suffices to recover the plaintext  $x\mathbf{a} + \mathbf{b}$  since the encryption internally employs a suitable error-correcting code. Indeed, [5] show how to combine a fast pseudorandom matrix with a linear-time error-correcting code and derive a linear-time encodable code that is pseudorandom under random noise but can be decoded in linear-time in the presence of random erasures.

$(\mathbf{d}_i, \mathbf{z}_i)$ , and Bob has to choose whether to learn  $\mathbf{d}_i$  or  $\mathbf{z}_i$ . For a selection vector  $I$ , Bob learns the vectors  $(\mathbf{d}_i)_{i \in I}$  and  $(\mathbf{z}_i)_{i \notin I}$ . By design, the latter vector reveals the secret  $s$  if and only if  $I$  satisfies the (\*) condition. Thus the CDS protocol effectively limits the query access to the OT's, and turns it into so-called a *generalized OT* (GOT) [29, 35, 49, 50].<sup>9</sup> Below, we present such a CDS protocol that achieves a constant computational overhead for both the sender and the receiver and information-theoretic security. Given such a protocol, we can immunize the RVOLE protocol by letting Alice re-encrypt her ciphertext  $\mathbf{d}$  under the secret  $s$ . This approach yields only computational security since the key  $s$ , which is a single field element, is shorter than the vector  $\mathbf{d}$ . (A CDS with longer secrets would lead to a super-constant computational overhead.) To achieve information-theoretic security, we note that it suffices to use  $s$  to encrypt the scalar  $x$  of the RVOLE protocol. That is, Alice invokes the modified RVOLE protocol (with the CDS mechanism) over the inputs  $x + s$  and  $\mathbf{b}$ . We show that if Bob's selection strategy  $I$  satisfies the (\*) condition, we can extract his inputs and perfectly simulate his view based on  $x\mathbf{a} + \mathbf{b}$ . If Bob's strategy  $I$  does not satisfy (\*), he learns the vector  $\mathbf{b}$  but  $x$  remains completely hidden, and we can perfectly simulate his view by sending  $\mathbf{a} = 0^w$  to the ideal RVOLE functionality. We refer to the resulting protocol as the *modified-RVOLE* protocol (See Sect. 5).

*Constructing the CDS.* Our construction of the CDS is linear-algebraic in nature. As a starting point, we employ the following standard fact: Fix a matrix  $T$  and a vector  $\mathbf{c}$ . Suppose that we “encrypt” a secret  $s$  by sampling a random row vector  $\mathbf{z}$  in the co-kernel of  $T$ , and publishing the “ciphertext”  $s + \langle \mathbf{z}, \mathbf{c} \rangle$ . If  $\mathbf{c}$  is spanned by  $T$  the ciphertext equals to  $s$ , on the other hand, if  $\mathbf{c}$  is *not* spanned by  $T$ , the ciphertext information theoretically hides  $s$ . Thus, *linear independence* is translated into *secrecy*. We can extend this idea to the CDS setting where we wish to reveal the secret iff  $\mathbf{c}[I]$  is in the span of  $T[I]$  for a subset  $I$ . To do this we reveal, for each  $i \notin I$ , the  $i$ th entry of our randomizer,  $\mathbf{z}_i$ , and send the ciphertext  $s + \langle \mathbf{z}, \mathbf{c} \rangle$  in the clear. Given this information, one can map the ciphertext to  $s + \langle \mathbf{z}[I], \mathbf{c}[I] \rangle$  which is decryptable if and only if  $\mathbf{c}[I] \in \text{span}(T[I])$ . While the above construction achieves privacy and correctness, it is not clear whether it achieves constant computational overhead. Indeed, we do not know how to sample a random vector in the co-kernel of  $T$  in linear time. Fortunately, there is a simple fix. To achieve a linear-time construction, we uniformly sample  $\mathbf{z}$  from the entire space (without limiting to the co-kernel) and append to the CDS the value  $\mathbf{z} \cdot T$  as a public value. Since right multiplication in  $T$  can be done in linear time, we can also left-multiply by  $T$  in linear time (following the “generalized transposition principle” [16, 38]) and so this variant can be realized with constant computational overhead. It is not hard to show that correctness

<sup>9</sup> GOT allows Bob to retrieve a subset of the messages of Alice that are “authorized” according to some predicate  $P$ . Previous constructions were either based on decomposable randomized encoding (aka private-simultaneous messages protocols) [35] or on secret-sharing [29, 49, 50]. We generalize these approaches by using CDS which is strictly weaker than both primitives.

and privacy still hold. (See the full version [10] for a formal definition of CDS and for details about the construction.)

*Securing the Protocol Against Active Alice?* We move on and consider an actively-corrupted Alice. Clearly, even if Alice deviates from the protocol and does not compute the vector  $\mathbf{d}$  properly, her view is still simulatable since all that she sees is a semantically-secure ciphertext  $\mathbf{c}$ . However, such misbehavior may lead Bob to abort and it is not fully clear how to simulate this case. Specifically, let us assume that Alice misbehaves and generates a vector  $\mathbf{d} \notin \text{colspan}(T|\mathbf{c})$ . The simulator detects this and can send an “abort” to the ideal functionality. However, in the real execution, Bob aborts only if his  $I$ -partial view is inconsistent, namely, if  $\mathbf{d}[I] \notin \text{colspan}((T|\mathbf{c})[I])$  where  $I = I(\mathbf{e})$  is the set of non-noisy coordinates in  $\mathbf{e}$ . To make the problem concrete, consider a malicious Alice that honestly computes  $\mathbf{d}$  and then adds noise to the first coordinate of  $\mathbf{d}$ . In this case, the above simulator sends an abort, but in the real protocol, Bob aborts only if the first coordinate is in  $I(\mathbf{e})$  which happens with constant probability  $1 - \mu$ . We present several solutions to this problem with different levels of efficiency.

1. The first solution is heuristic: We simply assume that the protocol is secure as it is. As evidence, we can prove this statement for the original RVOLE protocol (without the CDS) based on a variant of the Correlated Noisy-Codeword Hardness Assumption (See the full version [10]). By using a straightforward reduction from VOLE to RVOLE [5], this leads to the VOLE1 protocol (See Sect. 5.)
2. In the second solution, we first employ the modified-RVOLE protocol over random vectors  $\mathbf{a}'$  and  $\mathbf{b}'$  (this guarantees the ability to perfectly simulate an “abort” event), then use a small sub-protocol in which Alice proves that her CDS secret is independent of Bob’s input (based on a simple commitment), and finally, we shift the vectors back to the real inputs vectors  $\mathbf{a}$  and  $\mathbf{b}$  by exploiting the linearity of the VOLE functionality. To prove security we still need to extract Alice’s input in the event that the protocol does not abort. For this, we rely on the aforementioned “Correlated Noisy-Codeword” intractability assumption. In a nutshell, we show that under this assumption, the simulator who is given a malformed ciphertext  $\mathbf{d}$  either identifies that Bob would abort in the real execution or successfully extracts an effective input for Alice. Thus the security of the protocol can be based on the Correlated Noisy-Codeword assumption and on the fast pseudorandom matrix assumption. We refer to the resulting protocol as VOLE2 and note that it adds only a minor computational and communication overhead over RVOLE1 (See Sect. 6).
3. Finally, our most conservative solution (VOLE3) relies solely on the fast pseudorandom matrix assumption. The starting point is again the modified-RVOLE protocol. We begin by observing that there exist efficient tests that determine whether Alice’s ciphertext  $\mathbf{d}$  is “valid” and whether Alice’s vector of CDS messages  $\mathbf{z}$  is “valid”. Furthermore, when  $\mathbf{d}$  and  $\mathbf{z}$  are both valid, we can extract a unique effective input for Alice and properly simulate the protocol. We also note that there exists a strategy for Bob that detects (with

probability 0.5) whether Alice cheats. Indeed, in the OT phase Bob can toss a coin and ask with probability 1/2 to receive the vector  $\mathbf{d}$  (by using  $I = 1^m$ ) and with probability 1/2 the vector  $\mathbf{z}$  (by using  $I = 0^m$ ) and check validity. When running in this “detection mode” Bob effectively gives up on the computation and just verifies whether Alice misbehaves or not. Note that Bob’s decision is taken only in the OT phase and is hidden from Alice, and so effectively Alice first “commits” to strategy (cheat or not), and only then Bob decides whether to “call her bluff”. Furthermore, even when Bob acts as a detector, we can fully simulate his view (since the protocol is actively-secure against any deviation of Bob). We will exploit these observations to obtain a “silent” cut-and-choose version of the protocol. Specifically, we realize  $W$ -width VOLE based on many calls to the modified-RVOLE protocol over shorter vectors of width  $w \ll W$ . Ignoring some technical details, we “sacrifice” a small fraction of these calls for cheating-detection<sup>10</sup>, and glue together the remaining copies via a linear-time computable linear exposure-resilient function [20] (also known as perfect deterministic extractors for bit-fixing sources). Such functions can be constructed based on linear-time encodable codes (See Sect. 7).

## 2.4 Technical Overview of the Batch-OLE Protocol

The ADINZ passive batch-OLE protocol relies on an arithmetic analog of Beaver’s OT extension [14]. Given an arithmetic PRG  $G : \mathbb{F}^k \rightarrow \mathbb{F}^n$ , the idea is to realize a pseudorandom batch-OLE in which Alice holds the vectors  $\mathbf{c}$  and  $\mathbf{d}$  of length  $n$ , Bob holds a seed of a PRG  $\mathbf{x}$  of length  $k$  and the functionality stretches the seed  $\mathbf{x}$  to a pseudorandom vector  $\mathbf{y} = G(\mathbf{x})$  of length  $n$ , and delivers to Bob the value  $\mathbf{y} \odot \mathbf{c} + \mathbf{d}$  where  $\odot$  stands for entry-wise multiplication. The latter functionality  $f_G$  is realized by using an arithmetic variant of Yao’s protocol. Specifically, Alice prepares an arithmetic decomposable affine randomized encoding (DARE) [6, 36] (aka arithmetic garbled-circuit) of  $f_G$  and sends the “keys” that correspond to her entries. Bob recovers the keys of his inputs by making  $k$  calls to VOLE of width  $w = O(n/k)$  and recovers the output. When the PRG is computable in  $\mathbf{NC}^0$  the protocol can be realized with constant computational overhead. Clearly, the protocol is insecure in the presence of an actively corrupted Alice who can send a malformed encoding that corresponds to a different function. This well-known problem is extensively studied in the binary setting. We note that our concrete setting admits a simple and highly efficient solution.

Specifically, we strongly exploit the following properties: (1) We only care about the case where Bob’s input  $\mathbf{x}$  is chosen at random; (2) When Bob decodes the, possibly malformed, DARE (or the garbled circuit), each output of the computation can be written as a low-degree polynomial whose degree corresponds to

<sup>10</sup> Interestingly, this detection is performed “silently”: To test a session Bob just plays this session in a “detection mode”. In contrast, in typical cut-and-choose-based solutions, Bob asks Alice to “open” a session. In fact, in our protocol we can even hide from Alice which sessions were tested by Bob.



the degree of  $f_G$  which is very small (constant) compared to the field size  $|\mathbb{F}|$ .<sup>11</sup> (3) The function  $f_G$  is linear in Alice's inputs.

Equipped with these observations, we run an extended variant of the passively-secure protocol in which Alice holds the pair  $(\mathbf{c}, \mathbf{d})$  and another random pair of vectors  $(\mathbf{c}', \mathbf{d}')$  of similar length, and Bob learns  $\mathbf{y} \odot \mathbf{c} + \mathbf{d}$  and  $\mathbf{y} \odot \mathbf{c}' + \mathbf{d}'$ . Let us focus, for simplicity on the first output of  $f_G$ . By applying the decoder, Bob learns the values  $z_1$  and  $z'_1$  which are supposedly equal to  $\mathbf{y}_1 \cdot \mathbf{c}_1 + \mathbf{d}_1$  and to  $\mathbf{y}_1 \cdot \mathbf{c}'_1 + \mathbf{d}'_1$  where  $\mathbf{y}_1 = G(\mathbf{x})$ . Assuming that Alice behaves properly, Bob can now compute the value of  $\mathbf{y}_1 \cdot L(\mathbf{c}_1, \mathbf{c}'_1) + L(\mathbf{d}_1, \mathbf{d}'_1)$  for any linear combination  $L$ . The idea is to challenge Alice with a random non-trivial  $L$  and ask her to send  $c = L(\mathbf{c}_1, \mathbf{c}'_1)$  and  $d = L(\mathbf{d}_1, \mathbf{d}'_1)$ , and let Bob check whether  $L(z_1, z'_1) = cy_1 + d$ , and abort if the test fails.

First, observe that Alice's additional messages do not leak any information (since  $\mathbf{c}'_1$  and  $\mathbf{d}'_1$  mask the values of  $\mathbf{c}_1$  and  $\mathbf{d}_1$ ). Next, by using simple linear-algebraic arguments, we show that if Alice deviates from the protocol she will get caught except with probability  $O(D/|\mathbb{F}|)$  where  $D$  is the degree of the PRG  $G$ . To see this, assume that Alice sends malformed garbled circuits for Bob's first outputs of  $f_G$ . This means that Bob computes  $z_1 = Q(\mathbf{x})$  and  $z'_1 = Q'(\mathbf{x})$  for some degree- $D$  multivariate polynomials  $Q(\cdot)$  and  $Q'(\cdot)$  that are not both in the span of  $\{G_1(\cdot), 1\}$  (e.g., there are no scalars  $\mathbf{c}_1, \mathbf{d}_1$  for which  $Q(\mathbf{x}) = \mathbf{c}_1 G(\mathbf{x}) + \mathbf{d}_1$ ). Consequently, if we take a random linear combination  $L$  of  $Q(\cdot)$  and  $Q'(\cdot)$ , the resulting polynomial  $L(Q, Q')$  almost surely falls out of the span of  $\{G_1(\cdot), 1\}$ . In this case, no matter how the scalars  $c, d$  are chosen by Alice, the polynomial  $L(Q(\cdot), Q'(\cdot))$  will not be equal to the polynomial  $cG_1(\cdot) + d$ . Since both polynomials are of degree at most  $D$ , they will disagree over a random point  $\mathbf{x}$ , except with probability  $D/|\mathbb{F}|$ , and so Bob will almost surely catch the cheating (See Sect. 8 for more details).

As already mentioned this analysis crucially relies on the low-degree feature of the decoding procedure (property 2) to ensure that  $Q$  and  $Q'$  are of degree  $D$ . To the best of our knowledge, this is the first time that this feature is being employed.

## 3 Preliminaries

### 3.1 Linear Algebraic Notations

We define some linear-algebraic notation. Below  $\mathbb{F}$  denotes some finite field and  $m \in \mathbb{N}$  is a positive integer.

**Selective matrix-vector entries.** For a vector  $\mathbf{d} \in \mathbb{F}^m$  and a 0-1 vector  $I = (I_1, \dots, I_m) \in \{0, 1\}^m$ , we define the vector  $\mathbf{d}[I] \in \mathbb{F}^m$  such that its  $i$ th entry is  $d_i$  if  $I_i = 1$  and 0 otherwise. This notation can be used for both row and column vectors, and can be naturally extended to matrices as follows. For a

<sup>11</sup> Indeed, here we assume that the field is sufficiently large. In contrast, the VOLE1 and VOLE2 protocols can be realized over small fields as well.

$m \times k$  matrix  $M$  whose rows are denoted by  $M_1, \dots, M_m \in \mathbb{F}^k$ , and a binary vector  $I \in \{0, 1\}^m$ , we let  $M[I] \in \mathbb{F}^{m \times k}$  denote the matrix whose  $i$ th row is  $M_i$  if  $I_i = 1$  and  $0^k$  otherwise. Note that this operator is linear and can be written in the following matrix form:

$$\mathbf{d}[I] = \begin{pmatrix} I_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & I_m \end{pmatrix} \cdot \mathbf{d} \quad M[I] = \begin{pmatrix} I_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & I_m \end{pmatrix} \cdot M$$

**Vector of clean coordinates.** Let  $\mathbf{e} \in \mathbb{F}^m$  denote a vector (typically viewed as a noise vector). We define the vector  $I(\mathbf{e}) \in \{0, 1\}^m$  such that its  $i$ th entry is 1 iff  $e_i = 0$ . Note that by our notations  $\mathbf{e}[I(\mathbf{e})] = 0^m$ .

**Matrix-vector concatenation.** Let  $M$  be a matrix of dimensions  $m \times k$  and a vector  $\mathbf{c} \in \mathbb{F}^m$ . We define  $M|\mathbf{c}$  to be the result matrix that is obtained by concatenating the column vector  $\mathbf{c}$  to the matrix  $M$  from the right side.

**Bernoulli vector.** Let  $\text{BER}^m(p)$  for real number  $p \in [0, 1]$  be the distribution of binary vectors  $I = (I_1, \dots, I_m)$  of length  $m$  with i.i.d entries such that for any  $i$ :  $I_i$  takes a value of 1 with probability  $p$ .

*Family of Finite Fields.* We always assume that our functionalities are implicitly parameterized by a family of finite fields whose size may grow with the security parameter. Throughout the paper, we fix this family  $\mathbb{F} = \{\mathbb{F}_k\}_{k \in \mathbb{N}}$  and assume that it is *efficiently computable*, that is, one should be able to compute all field operations in  $\text{poly}(k)$  time (including the ability to add/subtract/multiply/divide and to sample a random field element). Note that this requirement implies that  $|\mathbb{F}_k| \leq 2^{\text{poly}(k)}$  and so field elements can be represented by  $\text{poly}(k)$ -bit strings. In fact, for our protocols, we only need black-box access to the field operations, and the ability to send field elements either directly or over an OT channel. By default, we also assume that the field is sufficiently large, e.g., exponentially large in the security parameter. For sufficiently large width parameter  $w$  (e.g., cubic in  $k$ ), our protocols for width- $w$  VOLE require  $O(w)$  field operations, and at most  $O(w \log |\mathbb{F}_k|)$  Boolean operations. Accordingly, the overall complexity is dominated by the arithmetic complexity  $O(w)$  which is optimal. Indeed, even in an insecure implementation,  $w$  arithmetic operations are needed for  $w$ -width VOLE.

It should be mentioned that the assumption regarding the field size is mainly needed for achieving linear-time efficiency and most of our protocols (or close variants of them) remain secure even when the field is of small constant-size (the error is always negligible in the security parameter). See Remark 2.

## 4 The ADINZ Protocol

The ADINZ [5] protocol for VOLE is based on a gadget (“encoder”) that allows fast encoding and decoding under erasures but semantically hides the encoded messages in the presence of noise. This gadget is mainly based on a public matrix

$M \in \mathbb{F}_k^{m \times k}$  with the following (LPN-style) pseudorandomness property: If we take a random vector in the image of  $M$ , and add a sparse noise to it, the resulting vector is computationally indistinguishable from a truly random vector over  $\mathbb{F}_k^{m(k)}$ . The noise distribution that is being used in the ADINZ protocol corresponds to an additive noise vector  $\mathbf{e} \in \mathbb{F}_k^{m(k)}$  where each coordinate of  $\mathbf{e}$  is assigned independently with the value of zero with probability  $1 - \mu$  and with a uniformly chosen non-zero element from  $\mathbb{F}_k$  with probability  $\mu$ . We let  $\mathcal{D}(\mathbb{F}_k)_\mu^m$  denote the corresponding noise distributions for such vectors of length  $m$ . For concreteness, the reader may think of  $\mu$  as a small constant, say  $1/4$ , however  $\mu$  can also be chosen so that it tends to 0 when the security parameter  $k$  tends to infinity. The properties of the ADINZ gadget are summarized in the following assumption.

**Assumption 3 (Fast pseudorandom matrix).** *There exists a noise rate  $\mu = \mu(k) < 1/2$  and an efficient randomized algorithm  $\mathcal{M}$  that given a security parameter  $1^k$  and a fields family representation  $\mathbb{F}$ , samples a  $m \times k$  ( $m = O(k^3)$ ) matrix  $M$  over  $\mathbb{F}_k$  such that the following holds:*

1. (Linear-time computation) *The mapping  $f_M : \mathbf{r} \rightarrow M\mathbf{r}$  can be computed in time that linear in the output length  $m$ , i.e., by performing  $O(m)$  arithmetic operations.*
2. (Noisy-codeword is pseudorandom) *The following ensembles are computationally indistinguishable:*

$$\{(M, M\mathbf{r} + \mathbf{e})\}_{k \in \mathbb{N}} \approx_c \{(M, \mathbf{u})\}_{k \in \mathbb{N}}$$

where  $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$ ,  $\mathbf{r} \leftarrow \mathbb{F}_k^k$ ,  $\mathbf{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$  and  $\mathbf{u} \leftarrow \mathbb{F}_k^m$ .

3. (Linear independence) *If we sample  $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$  and keep each of the first  $u = O(k \log^2 k)$  rows independently with probability  $1 - \mu$  (and remove all other rows), then, except with negligible probability in  $k$ , the resulting matrix has full rank of  $k$ .*

Concrete instantiations of this matrix-ensemble  $\mathcal{M}$  (e.g., based on sparse matrices or on the Druk-Ishai ensemble [25]) are discussed in [5]. The ADINZ encoder also makes use of a (non-cryptographic) linear error correcting code  $\text{Ecc} : \mathbb{F}_k^w \rightarrow \mathbb{F}_k^v$  which encodes vectors of length  $w$  into vectors of length  $v$  over the field  $\mathbb{F}_k$ , with constant rate  $R$  and linear time encoding and decoding, such that decoding is possible with high success probability from a constant fraction of erasures  $\mu'$  which is slightly larger than the noise rate  $\mu$ . (For  $\mu = \frac{1}{4}$  we can take  $\mu' = \frac{1}{3}$ ). Such codes are known to exist and can be efficiently constructed given a black-box access to  $\mathbb{F}_k$ . The code  $\text{Ecc}$  and the matrix  $M$  are combined together into the so-called protocol's encoder:

*ADINZ Protocol's Encoder.* Given  $k \in \mathbb{N}$ ,  $m = O(k^3)$ ,  $w = O(k^3)$ ,  $\mathbf{r} \in \mathbb{F}_k^k$  and  $\mathbf{a} \in \mathbb{F}_k^w$ , let  $M$  be a  $m \times k$  fast pseudorandom matrix. We define the encoding gadget  $E_r(\mathbf{a})$  to be:

$$E_r(\mathbf{a}) = M \cdot \mathbf{r} + 0^u \circ \text{Ecc}(\mathbf{a})$$

where  $\text{Ecc} : \mathbb{F}_k^w \rightarrow \mathbb{F}_k^v$ ,  $u = 2k \log^2 k$ ,  $v = m - u$  and  $\circ$  denotes concatenation (so  $0^u \circ \text{Ecc}(\mathbf{a})$  is a vector of length  $m$ ). Equivalently, for an information vector  $\mathbf{a} \in \mathbb{F}_k^w$  and randomness vector  $\mathbf{r} \in \mathbb{F}_k^k$ , we can write the encoder as

$$E_{\mathbf{r}}(\mathbf{a}) = T \cdot \begin{pmatrix} \mathbf{r} \\ \mathbf{a} \end{pmatrix},$$

where the encoder matrix is

$$T = \left( M_{m \times k} \left| \begin{array}{l} \mathbf{0}_{u \times w} \\ \text{Ecc}_{v \times w} \end{array} \right. \right) \tag{1}$$

and  $\text{Ecc}_{v \times w} \in \mathbb{F}_k^{v \times w}$  is the generating matrix of the error correcting code. By exploiting Assumption 3 and the features of the error correcting code, the encoder  $E$  satisfies the following properties:

1. (Fast and Linear) The mapping  $E_{\mathbf{r}}(\mathbf{a})$  can be computed by making only  $O(m)$  arithmetic operations. Moreover, it is a linear function of  $\mathbf{r}$  and  $\mathbf{a}$  and so  $E_{\mathbf{r}}(\mathbf{a}) + E_{\mathbf{r}'}(\mathbf{a}') = E_{\mathbf{r}+\mathbf{r}'}(\mathbf{a} + \mathbf{a}')$ .
2. (Hiding under errors) For any message  $\mathbf{a} \in \mathbb{F}_k^w$  and  $\mathbf{r} \leftarrow \mathbb{F}_k^k, \mathbf{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_m^\mu$  the vector  $E_{\mathbf{r}}(\mathbf{a}) + \mathbf{e}$  is pseudorandom. Namely: for any ensemble  $\{\mathbf{a}_k\}_{k \in \mathbb{N}}$  the following ensembles are computationally indistinguishable:

$$\{(M, E_{\mathbf{r}}(\mathbf{a}_k) + \mathbf{e})\}_{k \in \mathbb{N}} \approx_c \{(M, \mathbf{u})\}_{k \in \mathbb{N}}$$

where  $M \leftarrow \mathcal{M}(1^k, \mathbb{F}), \mathbf{r} \leftarrow \mathbb{F}_k^k, \mathbf{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$  and  $\mathbf{u} \leftarrow \mathbb{F}_k^m$ . In particular, a noisy codeword computationally “hides”  $\mathbf{a}$ .

3. (Fast decoding under erasures) Given a random vector  $I \leftarrow \text{BER}^m(1 - \mu)$  and a code  $\mathbf{d}[I] = E_{\mathbf{r}}(\mathbf{a})[I]$  (i.e. each coordinate is erased independently with probability  $\mu$ ) it is possible to recover the vector  $\mathbf{a}$ , with negligible error probability, by making only  $O(m)$  arithmetic operations. We first recover  $\mathbf{r}$  by solving the linear system  $\mathbf{d}_{\text{top}}[I_{\text{top}}] = M_{\text{top}}[I_{\text{top}}]\mathbf{r}$  (where “top” means top  $u$  coordinates) via Gaussian elimination in  $O(m)$  arithmetic operations. By Assumption 3 (property 3.) the system is likely to have a unique solution. Then we compute  $M[I_{\text{bot}}]\mathbf{r}$  in time  $O(m)$ , subtract from  $\mathbf{d}[I_{\text{bot}}]$  to get the vector  $\text{Ecc}(\mathbf{a})[I_{\text{bot}}]$  and recover  $\mathbf{a}$  by erasure decoding in time  $O(m)$ .

*Remark 2 (On the choice of parameters).* Some of the above requirements are tailored to achieve a VOLE of width  $w$  with an asymptotic computational complexity of  $O(w)$  field operations. This includes the choice of the values of  $m, w$ , and  $u$ , the requirements for the “fast” computation of  $E$  and “fast” decoding under erasures, and the assumption that the field size is exponential in the security parameter. All these requirements can be waived without affecting the security of the protocols. (Assuming that the pseudorandomness assumption holds.) In particular, for concrete settings, it may be better to set these parameters differently as done in our implementations (See the full version [10]).

In the full version [10] we show that the ADINZ protocol is vulnerable against actively corrupt receiver and prove that it is secure against an active sender under a new intractability assumption. (These parts will not be used in our subsequent protocols.)

## 5 RVOLE Protocol Against Actively-Corrupted Receiver

In this section, we construct a protocol for RVOLE, which is actively secure against Bob and passively secure against Alice. The protocol is based on the ADINZ protocol. We will later use this protocol as a building block of an actively secure VOLE protocol of width  $w$  over the field family  $\mathbb{F}$ . Our protocol relies on CDS for span membership. Formally, let  $f_{T,c} : \{0,1\}^m \rightarrow \{0,1\}$  be a predicate that receives a vector  $I \in \{0,1\}^m$  and accepts iff  $\mathbf{c}[I] \in \text{colspan}(T[I])$ . A CDS for  $f_{T,c}$  is a pair of algorithms  $\text{Enc}(I, S; R)$  and  $\text{Dec}(I, z)$  such that for an input  $I$ , secret  $S$  and randomizer  $R$  the “ciphertext”  $\mathbf{z} = \text{Enc}(I, S; R)$  perfectly hides  $S$  if  $f_{T,c}(I) = 0$ , and, otherwise,  $\text{Dec}(I, z)$  outputs  $S$ . In addition, the encoding function can be decomposed to an offline part that does not depend on  $I$  and  $m$  “online” parts each depending on a single bit of  $I$ , i.e.,  $\text{Enc}(I, S; R) = (\text{Enc}_0(S; R), \text{Enc}_1(I_1, S; R), \dots, \text{Enc}_k(I_m, S; R))$ . We also require that, both  $\text{Enc}(I, S; R)$  and  $\text{Dec}$ , are computable by  $O(m)$  arithmetic operations over  $\mathbb{F}$ . Construction of such a CDS with unconditional information-theoretic security appears in the full version [10].

**Protocol 4 (modified RVOLE protocol).** *To initialize the protocol Bob samples the matrix  $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$  and sends it to Alice.*

1. **Bob:** *Given an input  $\mathbf{a} \in \mathbb{F}_k^w$ , Bob samples vectors  $\mathbf{r} \leftarrow \mathbb{F}_k^k$  and  $\mathbf{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$ , sets  $I = I(\mathbf{e})$  and sends the vector:  $\mathbf{c} = E_{\mathbf{r}}(\mathbf{a}) + \mathbf{e}$  to Alice.*
2. **Alice:** *Given the inputs  $\mathbf{b} \in \mathbb{F}_k^w$ ,  $x \in \mathbb{F}_k$ , and Bob’s message  $\mathbf{c} \in \mathbb{F}_k^m$ , samples a random vector:  $\mathbf{r}' \leftarrow \mathbb{F}_k^k$  and a field element  $x' \leftarrow \mathbb{F}$  and computes the vector  $\mathbf{d} = x'\mathbf{c} + E_{\mathbf{r}'}(\mathbf{b})$ .*
3. **Alice:** *Samples randomness  $R$  for the span membership CDS and sets the secret  $\Delta = x - x'$ , and for each  $i \in [m]$  Alice computes two possible CDS messages  $z_{i,0} = \text{Enc}_i(0, \Delta; R)$  and  $z_{i,1} = \text{Enc}_i(1, \Delta; R)$ . In addition, Alice computes the CDS offline message by  $z_0 = \text{Enc}_0(\Delta; R)$  and sends  $z_0$  to Bob.*
4. **Alice and Bob:** *Invoke  $m$ -batch OT where the  $i$ th entry of Alice is the pair*

$$(z_{i,1}, d_i) \quad \text{and} \quad z_{i,0}$$

*and Bob uses the vector  $I$  as its selection vector.*

5. **Bob:**
  - *Collects all the  $z$ -part of the OT messages into a vector  $\mathbf{z} = (z_0, (z_{i,I_i})_{i \in [m]})$ , and applies the CDS decoder to recover the CDS secret  $\Delta = \text{Dec}(I, \mathbf{z})$ . If decoding fails Bob aborts.*
  - *If  $\mathbf{d}[I]$  is not in  $\text{colspan}((T|\mathbf{c})[I])$ , Bob aborts. Otherwise, Bob employs the decoding-under-erasures property of the gadget  $E$  (property 3.), computes the vector  $\mathbf{v}'$  (supposedly  $x'\mathbf{a} + \mathbf{b}$ ), shifts it by  $\Delta\mathbf{a}$  and outputs the result  $\mathbf{v} = \mathbf{v}' + \Delta\mathbf{a}$  (supposedly,  $x\mathbf{a} + \mathbf{b}$ ).*

The original ADINZ protocol is obtained by removing the blue parts and setting  $x' = x$  and outputting  $v'$ . As always, we assume the existence of an ideal  $m$ -batch OT channel. Through the analysis of Protocol 4, we assume that all the protocol's length parameters:  $m, w, v$  and  $u$  are polynomial functions of the security parameter  $k$ .

*Remark 3 (About the set-up).* In this protocol (and all the subsequent ones) the set-up step in which the matrix is sampled can be done once and for all. This is reflected in the security proofs which work even if the simulators receive  $M$  as an external input.

**Lemma 1.** *Under Assumption 3, the Protocol 4 realizes the RVOLE functionality of width  $w$  over  $\mathbb{F}$  in the OT-hybrid model with arithmetic complexity of  $O(w)$  (ignoring the initialization cost) and with the following guarantees:*

1. *Computational security against a passive adversary that corrupts Alice.*
2. *Computational privacy against an active adversary that corrupts Alice.*
3. *Perfect security against a passive adversary that corrupts Bob.*
4. *Perfect security against an active adversary that corrupts Bob and deviates from the protocol.*

*Proof (sketch).* The complexity bound follows from the complexity of the ADINZ encoder and from the complexity of the CDS encoder and decoder. One can easily verify that correctness holds when both parties are honest and that Alice's only incoming message is pseudorandom and is therefore simulatable regardless of Alice's behavior. This implies items 1 and 2. To simulate Bob, we collect  $c$  and  $I$  as chosen by (a possibly malicious) Bob, and check if  $c[I]$  is in  $\text{colspan}(T[I])$ . If the check passes we can extract Bob's effective input  $a'$  by solving the linear system  $c[I] = T[I] \cdot \begin{pmatrix} r \\ a' \end{pmatrix}$ , and if the check fails we set  $a'$  to be the all-zero vector.

Given  $v = xa' + b$  from the ideal functionality, we generate the CDS message of Alice just like in the real protocol by using some  $\hat{x}$  and  $\hat{b}$  that are consistent with the output  $v$ . It can be shown that this simulator perfectly emulates the real distribution. (See the full version [10] for details.)  $\square$

Some comments are in place:

1. (Computationally-unbounded Bob) The information-theoretic security against Bob holds even if the ideal OT channel is replaced with an OT protocol that provides statistical privacy for the sender (e.g., [1, 45]).
2. (Full security against active Alice) We do not know if Protocol 4 provides full security against an actively corrupt Alice and leave this as an open question. It seems reasonable to assume that the protocol achieves full security. Under this assumption, one can plug Protocol 4 to the standard RVOLE-to-VOLE transformation [5] and derive an actively-secure VOLE protocol. We refer to this protocol as *VOLE1*.

3. (Concrete communication complexity of CDS) Our concrete CDS communicates  $n + 1 = k + w + 1$  field elements in the offline message  $z_0$  and leaves the 1-messages  $z_{i,1}$  empty. Accordingly, each of the OT messages is just a single field element. Moreover, by resorting to computationally-private CDS (and exploiting PRGs), we can use an economic variant of the CDS in which each of the 0-messages,  $z_1, \dots, z_m$ , is of length  $k$  independently of the field size. As a result, the total communication complexity of the OT messages can be reduced to  $m \log |\mathbb{F}_k| + m \cdot k$ . Furthermore, this can be done while keeping the computational complexity linear.<sup>12</sup>
4. (On the achievable rate) Based on the aforementioned optimized CDS, Protocol 4 communicates  $m$  field elements from Bob to Alice,  $n + 1 = k + w$  field elements from Alice to Bob in the offline CDS message, and  $m$  field elements plus  $O(mk)$  bits over the OT-channel. Overall, the number of field elements that are communicated is  $2m + n + 1 = 2(u + v) + w + k + 1 = (2v + w)(1 + o(1))$  where the last equality holds since  $k = o(w)$  and  $u = o(v)$ . Recall that  $v$  is the length of the code produced by Ecc, which needs to be at least approximately  $\frac{1}{1-\mu}w$  to allow successful decoding of  $w$  field elements values from a noisy codeword with a fraction of  $\mu$  random erasures. Therefore, the communication rate of the protocol, measured as the length of the protocol's output  $w$ , divided by the communication complexity, approaches to:
 
$$\frac{w}{2v + w} = \frac{1 - \mu}{3 - \mu}.$$

If Assumption 3 holds for any constant error rate  $\mu > 0$  then we can obtain a rate approaching  $\frac{1}{3} - \varepsilon$  for any constant  $\varepsilon > 0$ . Furthermore, by choosing a non-constant erasure fraction of  $\mu = \frac{1}{f(k)}$  for  $f(k)$  that tends to infinity with  $k$  (for example  $f(k) = \frac{1}{\log k}$ ) we get an asymptotic rate of  $1/3$ , namely, in order to realize an RVOLE functionally of size  $w$  by our protocol  $3w$  fields elements should be communicated (where  $w$  and  $k$  tend to infinity).<sup>13</sup> The reduction to VOLE increases the communication by  $w$  additional field elements and so the rate of the VOLE1 protocol is  $\frac{1-\mu}{2(2-\mu)}$  which approaches to  $1/4$  for a small noise rate. Recall that the communication rate of the passively-secure ADINZ VOLE protocol approaches  $1/3$ .

## 6 Actively-Secure VOLE Under Correlated Noisy-Codewords

In this section, we realize the VOLE functionality directly while achieving active security against both Alice and Bob. For this, we introduce an additional, new,

<sup>12</sup> The computational complexity and communication complexity of batch-OT are measured as the total bit-length of the sent messages; see the full version [10] for a justification for this convention.

<sup>13</sup> In the context of binary codes, LPN-style assumptions with sub-constant  $\mu$  are quite standard.

“Correlated Noisy-Codeword” intractability assumption. We will also have to slightly modify the parameters of the ADINZ encoding matrix. Recall that the ADINZ encoding matrix  $T$  is defined as follows:

$$T = \left( M_{m \times k} \left| \begin{array}{l} \mathbf{0}_{u \times w} \\ \text{Ecc}_{v \times w} \end{array} \right. \right),$$

where  $m = \Omega(k^3)$ ,  $u = \Omega(k \log^2 k)$  and  $v = O(m)$ . For technical reasons we will need to slightly strengthen the linear-independence requirements of the matrix  $T$  as follows. Except with negligible probability over the choice of  $M$  and  $\text{Ecc}$  it must hold that: (a) If we sample a random subset of the first  $u$  rows of  $M$  by taking each row independently with probability  $1/\log^{1.5} k$  then the resulting matrix has full rank (all the columns are linearly independent); (b) The error-correcting code  $\text{Ecc}$  can correct up to  $O(\log^{1.1} k)$  errors and, as before, can recover from say  $1.2\mu v$  arbitrary erasures. (The constant 1.2 can be replaced with any constant larger than 1.).

## 6.1 The Correlated Noisy-Codeword Hardness Assumption

The following intractability assumption intuitively asserts that given a noisy codeword  $\mathbf{c} = T\mathbf{v} + \mathbf{e}$  of  $T$ , it is hard to efficiently generate a new noisy codeword  $\mathbf{d} = T\mathbf{v}' + \mathbf{e}'$  whose noise is non-trivially correlated with  $\mathbf{e}$  in the following sense. The new noise vector  $\mathbf{e}'$  agrees with the original noise vector  $\mathbf{e}$  with respect to the set of non-noisy coordinates  $I = I(\mathbf{e})$ , i.e.,  $\mathbf{d}[I] \in \text{colspan}(T[I])$ , but  $\mathbf{e}'$  is “far” from being a scalar multiple of  $\mathbf{e}$ . That is,  $\rho(\mathbf{d}, \text{colspan}(T|\mathbf{c})) = \ell$  where  $\rho(\mathbf{d}, S)$  is the minimal Hamming distance between a vector  $\mathbf{d}$  and a set of vectors  $S \subset \mathbb{F}^m$ . Observe that such a noisy codeword can be generated by sampling a vector in the column span of  $(T|\mathbf{c})$  and then modifying  $\ell$  entries with the hope that all these entries fall out of the set of clean coordinates  $I$ . Such an attack succeeds with probability  $\mu^\ell$ , the following assumption states that this is essentially the best that one can hope for up to polynomial speed-ups.<sup>14</sup>

**Assumption 5 (Correlated noisy codeword).** *For a distribution  $\mathcal{T}$  over matrices in  $\mathbb{F}_k^{m \times n}$  where  $m(k), n(k)$  are some polynomials in the security parameter  $k$ , and for a constant noise rate of  $\mu < 1/2$ , the Correlated Noisy-Codeword assumption asserts that for every efficient adversary  $A^*$  there exists some negligible  $\varepsilon(k)$  and constant  $C$  such that for every integer  $\ell \leq m$ :*

$$\Pr_{\mathbf{d} \leftarrow A^*(\mathbf{c})} [\mathbf{d}[I] \in \text{colspan}(T[I]) \quad \wedge \quad \rho(\mathbf{d}, \text{colspan}(T|\mathbf{c})) = \ell] \leq \exp(-C\ell) + \varepsilon(k)$$

where  $T \leftarrow \mathcal{T}$  and  $\mathbf{c} = T\mathbf{v} + \mathbf{e}$  for  $\mathbf{v} \leftarrow \mathbb{F}_k^n$ ,  $\mathbf{e} \leftarrow \mathcal{D}(\mathbb{F}_k)_\mu^m$  and  $I = I(\mathbf{e})$ .

<sup>14</sup> The concrete formulation that is taken here is chosen for the sake of simplicity. More refined and conservative versions (e.g., that assume better speed-ups and consider sub-constant noise regimes) can be adopted as well.



For our purposes, it suffices to assume the “super-logarithmic version of Assumption 5” that asserts that for every super-logarithmic function  $\ell(k) = \omega(\log k)$  the success probability in the above game is negligible in  $k$ . (Note that this variant follows from the above formulation.) We conjecture that every matrix distribution whose noisy codewords are pseudorandom also satisfies this assumption, and provide some evidence for this in the full version [10]. From now, we will always use the super-logarithmic version of the assumption with respect to the distribution  $\mathcal{T}$  that corresponds to the ADINZ encoding matrix.

We will make use of the following simple observation whose proof is deferred to the full version [10].

**Lemma 2.** *There exists a probabilistic polynomial-time algorithm  $G$  that given the matrix  $T$  and the vectors  $\mathbf{c}, \mathbf{d} \in \mathbb{F}_k^m$  outputs a vector  $\mathbf{u} \in \mathbb{F}_k^{n+1}$  with the following guarantee. Except with negligible probability in  $k$  over the choice of  $(T, \mathbf{c})$  (which are distributed as in Assumption 5) and the randomness of  $G$ , if  $\mathbf{d}$  is  $\ell$ -close to  $\text{colspan}(T|\mathbf{c})$  for  $\ell = O(\log^{1.1} k)$  then the algorithm outputs  $\mathbf{u}$  such that  $(T|\mathbf{c}) \cdot \mathbf{u}$  is  $\ell$ -close to  $\mathbf{d}$ .*

## 6.2 The VOLE2 Protocol

We present our VOLE protocol and prove security under Assumption 5. The protocol employs an ideal-commitment functionality, aka *commitment channel*, which is a 2-phase ideal functionality of the following form. In the commit phase, the functionality takes an input  $x$  from a sender (e.g., a field element), and delivers a commit message to the receiver. At a later phase, the sender can de-commit by sending an “open” message to the functionality which delivers to the receiver the committed message  $x$ . Such an ideal commitment channel can be constructed based on OT-channel [21, 22] perfect security against the receiver and statistical security against the sender the communication and computational complexity of  $k$  OT-calls and  $k$  field additions (where  $k$  is the statistical security parameter).

**Protocol 6 (VOLE2 protocol).** *To initialize the protocol Bob samples the matrix  $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$  and sends it to Alice.*

1. **Alice and Bob:** Hold inputs  $x \in \mathbb{F}_k$  and  $\mathbf{a}, \mathbf{b} \in \mathbb{F}_k^w$  respectively. The parties invoke Protocol 4 for RVOLE where Bob’s input is a random vector  $\mathbf{a}' \leftarrow \mathbb{F}_k^w$ , and Alice’s input is  $x$  and a random vector  $\mathbf{b}' \leftarrow \mathbb{F}_k^w$ . Let  $x'$  denote the random field element that is being sampled by Alice in the protocol and let  $\Delta_x = x - x'$  denote the secret that Alice delivers via the CDS.
2. **Alice:** Sends  $\Delta_x$  over an ideal commitment channel which delivers a “commit” message to Bob.
3. **Bob:** If Bob aborts during Protocol 4 then he aborts the entire execution. Otherwise, Bob recovers from the protocol the vector  $\mathbf{v}'$  (supposedly  $x'\mathbf{a}' + \mathbf{b}'$ ) and the CDS secret  $\Delta_x$  (supposedly  $x - x'$ ). Bob sends  $\Delta_x$  to Alice.
4. **Alice:** Verifies that Bob’s message equals to  $\Delta_x$ , and aborts if the check fails. If the check passes, Alice decommits by sending an “open” message to the commitment channel which delivers the committed value,  $\Delta_x$  to Bob.

5. **Bob:** Verifies that the decommitted value equals to  $\Delta_x$ , and aborts if the check fails. If the check passes, Bob sends to Alice the vectors  $\mathbf{v} = \mathbf{v}' + \Delta_x \mathbf{a}' + \mathbf{b}$  (supposedly,  $x\mathbf{a}' + \mathbf{b}' + \mathbf{b}$ ) and  $\Delta_{\mathbf{a}} = \mathbf{a} - \mathbf{a}'$ .
6. **Alice:** Computes the vector  $\mathbf{w} = x\Delta_{\mathbf{a}} + \mathbf{v} - \mathbf{b}'$  (supposedly,  $x\mathbf{a} + \mathbf{b}$ ) and outputs the result.

In the full version [10] we prove that the protocol is computationally secure against an actively corrupt Alice, and statistically secure against an actively corrupt Bob. By replacing the commitment with  $k$  calls to OT, we derive the following lemma.

**Lemma 3.** *Suppose that Assumptions 3 and 5 hold. Then protocol 6 for honest parties realizes the VOLE functionality of width  $w$  over  $\mathbb{F}$  with arithmetic complexity of  $O(w)$  (ignoring the initialization cost) in the OT-hybrid model. The protocol is statistically-secure against an active sender Bob with negligible deviation error and computationally secure against an active receiver Alice.*

Some comments are in place:

1. (Unbounded sender) Here too, the protocol achieves information-theoretic security against the sender even if the ideal channels are replaced by an OT protocol that provides statistical privacy for the sender and by a commitment scheme that is statistically hiding. (The latter reduces to the former by using the OT-to-Commitment transformation.
2. (Working over small fields) The statistical error is exponentially-small in the bit-length of the field element  $\Delta_x$ . (This essentially corresponds to the case where Bob guesses the value of  $\Delta_x$  despite playing dishonestly in the RVOLE protocol in a way that keeps the CDS secret hidden). Thus, when the field is small the error is only  $1/|\mathbb{F}_k|$ . Nevertheless, even when the field is small, one can easily get a negligible error at a minor cost by randomly padding the element  $\Delta_x$  to length  $k$ .
3. (On the achievable rate of Protocol 6) The communication of Protocol 6 (VOLE2) consists of  $(2v+w)(1+o(1))$  field elements in Step 1 (when invoking the RVOLE Protocol),  $2k$  field elements to commit (via  $k$  OT calls) and to de-commit, and additional  $2w + 1$  elements. Since  $k = o(w)$ , the total communication complexity is  $(2v + 3w)(1 + o(1))$ . Recall that  $v$  is the length of the code produced by Ecc, which needs to be at least approximately  $\frac{1}{1-\mu}w$  to allow successful decoding of  $w$  field elements values from a noisy codeword with fraction of  $\mu$  random erasures. Therefore, the communication rate of the protocol, measured as the length of the protocol's output  $w$ , divided by the communication complexity, approaches to

$$\frac{w}{2v + 3w} = \frac{1 - \mu}{5 - 3\mu}.$$

If Assumption 3 holds for any constant error rate  $\mu > 0$  then the rate of RVOLE2 approaches to  $\frac{1}{5} - \varepsilon$  for any constant  $\varepsilon > 0$ .

4. (Comparison to VOLE1) In terms of communication we pay an amortized cost of an extra field element per each VOLE entry compared to VOLE1, which, in turn, pays an extra field element compared to the passively-secure ADINZ protocol. In terms of computation, VOLE2 has a negligible overhead compared to VOLE1 which consists of a single commitment (for the entire VOLE), and, an amortized cost of  $1/R$  field multiplication and  $2/R$  field additions per VOLE entry where  $R = w/v$  is the rate of the error-correcting code.<sup>15</sup>

## 7 Actively-Secure VOLE Under Fast Pseudorandom Matrix

In this section, we describe a VOLE protocol with full active security based on the modified-RVOLE protocol (Protocol 4). Following the outline in Sect. 2.3, we begin with some useful observations.

### 7.1 Useful Observations

*More CDS Properties.* We will make use of the fact that our concrete CDS sends messages only on “zero” inputs. (That is, our CDS is effectively a secret sharing scheme for the negated predicate.) Let  $\mathbf{z} = (z_0, z_{1,0}, \dots, z_{m,0})$  be a (possibly malformed) full vector of CDS messages. We say that  $\mathbf{z}$  is *valid* if for every input  $I$  that satisfies the underlying predicate  $f$ , the CDS decoder recovers the same secret. We say that  $\mathbf{z}$  is *honestly generated* if it is generated by invoking the CDS message generator honestly on some random tape and some secret. (By perfect correctness, an honestly generated CDS is always valid.) We assume the existence of an efficient tester  $\mathcal{T}$  that rejects every invalid  $\mathbf{z}$ , and accepts every honestly generated  $\mathbf{z}$ . (That is  $\mathcal{T}$  is allowed to accept a vector that is not honestly generated as long as it is valid.) Furthermore, if  $\mathcal{T}$  accepts then it should be able to recover the secret. Our CDS construction satisfies these properties.

*Closer Look at Cheating Alice.* Fix some strategy  $A^*$  for a malicious Alice in Protocol 4. Formally, this is a deterministic mapping that takes Alice’s inputs  $(x, \mathbf{b})$ , the public matrix  $M$ , and Bob’s message  $\mathbf{c}$  and outputs a CDS message vector  $\mathbf{z} = (z_0, (z_{i,0})_{i \in [m]})$  and a vector  $\mathbf{d}$  (to be placed on the 1-inputs of the OT). If either  $\mathbf{z}$  is invalid or  $\mathbf{d}$  is invalid in the sense that  $\mathbf{d} \notin \text{colspan}((T|\mathbf{c}))$ , we say that  $A^*$  *cheats* on  $(x, \mathbf{b}, M, \mathbf{c})$ . In the full version [10] we show that when Alice does not cheat, her “effective input” can be extracted given her OT messages, and use this to prove the following lemma. Let us denote Protocol 4 by  $\Pi$ .

<sup>15</sup> Recall that VOLE1 is obtained by combining the RVOLE-to-VOLE transformation with Protocol 4 for RVOLE. Accordingly, the latter protocol achieves provable active security against the Sender, provable passive security against the Receiver, and heuristic active security against the Receiver.

**Lemma 4.** *There exists a simulator  $\text{SIM}'(x, \mathbf{b})$  that makes a black-box use of  $A^*$  and simulates  $\Pi$  whenever  $A^*$  does not cheat. Formally, for every sequence of inputs  $((x_k, \mathbf{b}_k), \mathbf{a}_k)$  it holds that the ensemble*

$$\left[ \text{REAL}_{A^*, \Pi}((x_k, \mathbf{b}_k), \mathbf{a}_k) \mid A^* \text{ doesn't cheat} \right]$$

*is computationally indistinguishable from the ensemble*

$$\left[ \text{IDEAL}_{\text{SIM}, f_k}(\mathbf{a}_k, \mathbf{b}_k) \mid \text{SIM doesn't fail} \right].$$

The next crucial observation is that there exists a strategy for Bob that detects (with probability 0.5) whether Alice cheats. Indeed, as explained in the introduction, in the OT phase Bob can toss a coin and ask with probability 1/2 to receive the vector  $\mathbf{d}$  (by using  $I = 1^m$ ) and with probability 1/2 the vector  $\mathbf{z}$  (by using  $I = 0^m$ ) and check validity. When running in this “detection mode” Bob effectively gives up on the computation and just verifies whether Alice misbehaves or not. Note that Bob’s decision is taken only in the OT phase and is hidden from Alice, and so effectively Alice first “commits” to strategy (cheat or not), and only then Bob decides whether to “call her bluff”. Furthermore, even when Bob acts as a detector, we can fully simulate his view (since the protocol is actively-secure against any deviation of Bob). We will exploit this property to obtain a “silent” cut-and-choose version of the protocol as follows.

## 7.2 The VOLE3 Protocol

Let  $\Pi$  denote Protocol 4 instantiated with width  $w(k) = O(k^3)$  and recall that  $\Pi$  makes a call to an  $m = m(k)$ -batch OT channel. The new protocol (hereafter denoted as VOLE3) realizes VOLE with width  $W = W(k)$  (for some value that will be determined later) by making  $t = t(k)$  calls to  $\Pi$ , and by “opening”  $p = p(k)$  sessions for detecting a potential cheating by Alice. In addition to these parameters, we let  $s = s(k) = t(k) - p(k)$  denote the number of remaining “un-opened” sessions, and let  $\ell = \ell(k)$  be a leakage parameter. The product  $\ell p/t$  should be polynomial in the security parameter  $k$  and, for efficiency purposes,  $s$  should be  $\Omega(t)$ . For example, set  $t = k$ ,  $p = \ell = k^{0.9}$  and  $s = k - k^{0.9}$ . Again, we make use of ideal commitments which can be realized based on OT channels.

*Linear-time resilient functions.* We will need a linear mapping  $\text{Ext} : \mathbb{F}^{sw} \rightarrow \mathbb{F}^{(1-\beta)sw}$  where  $\beta = \beta(k) < 1$  is bounded away from 1, with the following properties: (1)  $\text{Ext}$  should be computable in linear arithmetic time (i.e., by making  $O(sw)$  operations); and (2) The distribution  $\text{Ext}(X)$  should be uniform whenever the input  $X$  is uniform except for at most  $\ell' = \ell(k)w + 1 = o(sw)$  entries that may be arbitrarily fixed. Formally, for every  $\ell'$ -subset  $L \subset [sw]$  and fixing  $(X_i)_{i \in L} \in \mathbb{F}^{\ell'}$  if  $(X_i)_{i \notin L}$  is uniform over  $\mathbb{F}^{sw-\ell'}$ , the output  $\text{Ext}(X_1, \dots, X_{sw})$  is uniform over  $\mathbb{F}^{(1-\beta)sw}$ . Such functions are known as  $\ell'$ -resilient functions [20] and can also be viewed as perfect deterministic extractors for bit-fixing sources. One can realize such functions, with the desired parameters, based on linear-time

encodable error-correcting codes with rate  $1 - \beta$  and distance  $\ell' + 1$  (see [20] and [24, Theorem 3.1.7]). The width of VOLE3 is taken to be the output length of Ext, i.e.,  $W(k) = (1 - \beta)sw$ .

**Protocol 7 (VOLE3 Protocol).** Upon initialization, bob samples  $M \leftarrow \mathcal{M}(1^k, \mathbb{F})$  and sends it to Alice. The input of Bob is a pair of vectors  $\mathbf{g}, \mathbf{f} \in \mathbb{F}_k^W$  and the input of Alice is  $x \in \mathbb{F}_k$ .

1. **Bob:** Invokes  $t$  independent parallel sessions of  $\Pi$  with the matrix  $M$  as a public parameter, where the  $j$ th private input is a random vector  $\mathbf{a}_j \leftarrow \mathbb{F}_k^w$ . For each such session  $j \in [t]$ , Bob computes the first-round message  $\mathbf{c}_j$  as in Step 1 of  $\Pi$ , and sends  $\mathbf{c}_j$ . Let  $I_j$  denote the (vector representation of the) set of clean coordinates in  $\mathbf{c}_j$ .
2. **Alice:** Samples  $x^* \leftarrow \mathbb{F}_k$ . For every  $j \in [t]$ , Alice sets her inputs for the  $j$ th session to be  $(x_j, \mathbf{b}_j)$  where  $x_j = x^*$  and  $\mathbf{b}_j \leftarrow \mathbb{F}_k^w$ , she samples a random tape for the  $j$ th session, sends the corresponding offline CDS message  $z_{i,0}$  to Bob, and computes the vectors  $(\mathbf{z}_j, \mathbf{d}_j)$  that will be sent in the OT phase of the  $j$ th session (by following Steps 2 and 3 of  $\Pi$ ). Here we view Alice's input to the  $m$ -batch-OT as a pair of  $m$ -long vectors.
3. **OT-phase:** Bob samples a random  $p$ -subset  $P \subset [t]$  of sessions that will be "opened". For each  $j \in [t]$  in parallel, the parties invoke the  $m$ -batch OT channel of the  $j$ th session. Alice's input is  $(\mathbf{z}_j, \mathbf{d}_j)$ . If  $j \notin P$  Bob's input is  $I_j$ ; Otherwise, Bob samples a random bit  $\sigma_j \leftarrow \{0, 1\}$  and uses a trivial selection vector  $I'_j := \sigma_j^m \in \{0^m, 1^m\}$ .
4. **Bob:** If cheating is detected in one of the "opened copies"  $j \in P$  (i.e., if the received vector is invalid), Bob aborts. Otherwise, let  $S = [t] \setminus P$  denote the set of unopened copies. For every  $j \in S$ , Bob recovers the output  $\mathbf{v}_j \in \mathbb{F}_k^w$  of the  $j$ th session just like in Step 5. in  $\Pi$  (hereafter referred to as  $\Pi_5$ ). If the output is "abort" Bob sets  $\mathbf{v}_j = 0^w$ .
5. **Sub-protocol:** To verify that Alice's inputs  $(x_j)_{j \in S}$  are all equal, the parties do:
  - **Bob:** Computes the sum  $\delta$  of all the last elements of the vectors  $(\mathbf{a}_j \in \mathbb{F}_k^w)_{j \in S}$ , i.e.,  $\delta := \sum_{j \in S} \mathbf{a}_j[w]$  and sends to Alice the pair  $(S, \delta)$ . (Bob challenges Alice to compute the sum  $\sum_{j \in S} \mathbf{v}_j[w]$ .)
  - **Alice:** Sends  $\lambda = x^* \delta + \sum_{j \in S} \mathbf{b}_j[w]$  over the ideal commitment channel which delivers a "commit" message to Bob.
  - **Bob:** Given a "commit" message, sends the value  $\lambda' := \sum_{j \in S} \mathbf{v}_j[w]$ .
  - **Alice:** decommits by sending an "open" message to the commitment channel if  $\lambda' = \lambda$ , else Alice aborts.
  - **Bob:** Halts with an abort symbol if Alice does not open the commitment or if the decommitment  $\lambda \neq \lambda'$ , and continues otherwise.
6. **Alice:** Sends  $\Delta = x - x^*$ .
7. **Bob:** Aligns the results of the unopened sessions vectors by setting

$$\mathbf{u}_j := \mathbf{v}_j + \Delta \mathbf{a}_j, \quad \forall j \in S,$$

concatenates the vectors  $(\mathbf{a}_j)_{j \in S}$  to a single vector  $\mathbf{a}_S \in \mathbb{F}_k^{sw}$  and the vectors  $(\mathbf{u}_j)_{j \in S}$  to a single vector  $\mathbf{u}_S \in \mathbb{F}_k^{sw}$ , and extracts the vectors

$$\mathbf{a}' := \text{Ext}(\mathbf{a}_S), \quad \mathbf{u}' := \text{Ext}(\mathbf{u}_S).$$

Bob sends to Alice the vectors  $\boldsymbol{\alpha} := \mathbf{f} - \mathbf{a}'$  and  $\mathbf{h} := \mathbf{u}' + \mathbf{g}$ .

8. **Alice:** Concatenates  $(\mathbf{b}_j)_{j \in S}$  to a vector  $\mathbf{b}_S \in \mathbb{F}_k^{sw}$ , extracts  $\mathbf{b}' := \text{Ext}(\mathbf{b}_S)$ , and outputs  $\mathbf{h} + x\boldsymbol{\alpha} - \mathbf{b}'$ .

The protocol has an arithmetic complexity of  $O(tw) = O(sw) = O(W)$ , as required. The communication complexity is dominated by the complexity of Steps 1–3 and Step 7 which is  $t \cdot C_\Pi(w) + 2W = tO(w) + O(W) = O(W)$  where  $C_\Pi(w)$  is the communication complexity of  $\Pi$  over width  $w$ . (The communication in Steps 4–6 consists of  $O(s)$  bits and  $O(k)$  field elements for realizing the commitment channel via OT). By employing extractors that shrink their input by a factor of  $1 - \beta$  for arbitrarily small constant  $\beta$  (e.g., based on the codes of [33]), we can take  $W = (1 - \beta)(1 - o(1))tw$ . Recalling that  $C_\Pi(w)$  approaches to  $3w$ , the total communication of VOLE3 approaches to  $t3w + 2W = 5W/(1 - \beta - o(1))$ , i.e., the asymptotic rate approaches to  $1/5$ . The simulators for Alice and Bob appear in the full version [10], leading to Theorem 1.

## 8 Batch-OLE

Let  $n(k)$  be a polynomial in  $k$  and let  $G : \mathbb{F}^k \rightarrow \mathbb{F}^n$  be a function that can be computed by a constant-depth bounded fan-in arithmetic circuit (aka  $\text{NC}^0$  arithmetic circuit). We assume that  $G$  is *input-regular* in the sense that each input affects at most  $O(n/k)$  outputs. Let  $f_G : \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^k \rightarrow \mathbb{F}^n$  be the mapping

$$(\mathbf{c}, \mathbf{d}, \mathbf{x}) \mapsto G(\mathbf{x}) \odot \mathbf{c} + \mathbf{d},$$

where  $\mathbf{c}, \mathbf{d} \in \mathbb{F}^n$ ,  $\mathbf{x} \in \mathbb{F}^k$  and  $\odot$  stands for entry-wise multiplication. We will be interested in computing the *Generalized Affine Functionality*  $\mathcal{F}_G$  which takes the vectors  $\mathbf{c}, \mathbf{d} \in \mathbb{F}^n$  from Alice (the sender) and delivers to the receiver Bob a random vector  $\mathbf{x} \in \mathbb{F}^k$  together with the outcome of  $f_G(\mathbf{c}, \mathbf{d}, \mathbf{x})$ . The functionality  $\mathcal{F}_G$  is corruption-aware and it allows a malicious Bob to choose  $\mathbf{x}$  arbitrarily. Throughout, we assume, wlog, that, for every  $i \in [n]$ , the output of  $G_i(\cdot)$  is a non-constant function (i.e., for some  $x, x'$ , it holds that  $G_i(x) \neq G_i(x')$ ).

*Realizing  $\mathcal{F}_G$ .* In the passive setting, it is easy to realize  $\mathcal{F}_G$  by using an arithmetic variant of Yao's protocol [52] where the garbled circuit is replaced with fully-decomposable randomized encoding (DARE). (See the full version [10] for background on DARE; for now, the reader can think of DARE as arithmetic garbled circuit.) In fact, this protocol also provides active security against the receiver Bob. We will show how to cheaply upgrade the protocol and provide active security against the sender as well. Our protocol employs DARE for  $f = f_G$ . By [8, 37] there exists a DARE  $\hat{f}$  which can be encoded and decoded by an  $O(n)$ -size arithmetic circuit. Since  $\hat{f}$  is decomposable we can write it as

$$\hat{f}(\mathbf{x}, \mathbf{c}, \mathbf{d}; \mathbf{r}) = (f_0(\mathbf{c}, \mathbf{d}; \mathbf{r}), (\hat{f}_i(x_i; \mathbf{r}))_{i \in [k]}).$$

(That is, we collapse the  $\mathbf{c}$ -dependent outputs and the  $\mathbf{d}$ -dependent outputs to a single “block”.) Furthermore, for every  $i \in [k]$  the function  $\hat{f}_i(x_i; \mathbf{r})$  can be written as  $x_i \mathbf{a}_i + \mathbf{b}_i$  where the vectors  $(\mathbf{a}_i, \mathbf{b}_i)$  are sampled by a “key-sampling” mapping  $K_i : \mathbf{r} \mapsto (\mathbf{a}_i, \mathbf{b}_i)$ . By padding, we may assume that the key generation functions  $K_1, \dots, K_k$  have uniform output length of  $w$ , and since  $G$  is an input-regular  $\mathbf{NC}^0$  function, it holds that  $w = O(n/k)$ . Moreover, recall that given  $(\mathbf{r}, \mathbf{c}, \mathbf{d})$  we can collectively compute  $\hat{f}_0(\mathbf{r}, \mathbf{c}, \mathbf{d}), (K_i(\mathbf{r}))_{i \in [k]}$  by  $O(n)$  arithmetic operations. We denote the randomness complexity of the DARE by  $\rho$ . We realize  $\mathcal{F}_G$  by Protocol 8 which employs an ideal batch-VOLE of width  $2w$  and length  $k$  (that is,  $k$  parallel calls to VOLE of width  $2w$ ), and performs 2 additional rounds of interaction. In the full version [10] we prove the following lemma.

**Lemma 5.** *Protocol 8 realizes  $\mathcal{F}_G$  with information-theoretic active security in a constant number of rounds by making a single call to an ideal  $k$ -length  $O(n/k)$ -width VOLE, communicating  $O(n)$  field elements, and performing  $O(n)$  arithmetic operations. The protocol is perfectly secure against an active adversary that corrupts Bob (receiver) and statistically secure against an adversary that actively corrupts Alice (the sender) where the statistical deviation is  $O(D/|\mathbb{F}|) = \text{negl}(k)$  where  $D = O(1)$  is the degree of  $G$ .*

**Protocol 8 (GA Protocol).** *Let  $X$  denote the uniform distribution over  $\mathbb{F}^k$ . Given an input  $\mathbf{c}, \mathbf{d} \in \mathbb{F}^n$  for Alice and an empty input for Bob, the protocol proceeds as follows.*

1. **Alice:** selects randomness  $\mathbf{r} \leftarrow \mathbb{F}^\rho$  for the encoding, and sets  $\mathbf{v}_0 = \hat{f}_0(\mathbf{c}, \mathbf{d}; \mathbf{r})$  and  $(\mathbf{a}_i, \mathbf{b}_i) = K_i(\mathbf{r})$  for every  $i \in [k]$ . In addition, Alice samples random  $\mathbf{c}', \mathbf{d}' \in \mathbb{F}^n$  and  $\mathbf{r}' \leftarrow \mathbb{F}^\rho$ , and sets  $\mathbf{v}'_0 = \hat{f}_0(\mathbf{c}', \mathbf{d}'; \mathbf{r}')$  and  $(\mathbf{a}'_i, \mathbf{b}'_i) = K_i(\mathbf{r}')$  for every  $i \in [k]$ .
2. The parties invoke  $k$ -length of  $O(n/k)$ -width VOLE as follows. Bob samples  $\mathbf{x} \leftarrow X$ , and plays the role of the receiver with the input  $\mathbf{x} = (x_1, \dots, x_k)$ . Alice plays the role of the sender and sets her inputs to be the  $2w$ -length vectors  $(\mathbf{a}_i \circ \mathbf{a}'_i)$  and  $(\mathbf{b}_i \circ \mathbf{b}'_i)$  for every  $i \in [k]$  where  $\circ$  denotes concatenation. For every  $i \in [k]$ , the functionality delivers to Bob the vectors  $\mathbf{v}_i = x_i \mathbf{a}_i + \mathbf{b}_i$ ,  $\mathbf{v}'_i = x_i \mathbf{a}'_i + \mathbf{b}'_i$ .  
In addition, Alice sends to Bob the vectors  $\mathbf{v}_0$  and  $\mathbf{v}'_0$ .
3. **Bob:** decodes the vectors  $\mathbf{z}, \mathbf{z}' \in \mathbb{F}^n$  by setting  $\mathbf{z} = \text{Dec}((\mathbf{v}_i)_{0 \leq i \leq k})$  and  $\mathbf{z}' = \text{Dec}((\mathbf{v}'_i)_{0 \leq i \leq k})$  where  $\text{Dec}$  is the decoder of the DARE. In addition, Bob sends to Alice a random non-zero field element  $\alpha \leftarrow \mathbb{F}^*$ .
4. **Alice:** sends to Bob the  $n$ -length vectors  $\boldsymbol{\gamma} = \mathbf{c} + \alpha \mathbf{c}'$  and  $\boldsymbol{\delta} = \mathbf{d} + \alpha \mathbf{d}'$ .
5. **Bob:** outputs  $(\mathbf{x}, \mathbf{z})$  if  $\mathbf{z} + \alpha \mathbf{z}' = G(\mathbf{x}) \odot \boldsymbol{\gamma} + \boldsymbol{\delta}$ , and, otherwise, aborts.

*Constructing Batch-OLE.* One can easily construct batch-OLE of length  $n$  based on a single call to the  $\mathcal{F}_G$  functionality where  $G$  is a PRG (see [5]). If  $G : \mathbb{F}^k \rightarrow \mathbb{F}^n$  is an  $\mathbf{NC}^0$  PRG with, say quadratic stretch  $n = k^2$ , we can further realize  $\mathcal{F}_G$  using the above protocol with constant computational overhead. Note that the protocol assumes that  $G$  is input-regular. This requirement is satisfied by natural PRG candidates in  $\mathbf{NC}^0$ , and, in fact, it can be fully waived (see the full version [10]).

*On the Concrete Complexity of the Protocol.* The complexity of the protocol is dominated by the parameters of the PRG  $G$  and the cost of the DARE for  $f_G$ . Assuming that  $G : \mathbb{F}^k \rightarrow \mathbb{F}^n$  is a  $d$ -local regular PRG whose DARE can be computed and decoded by  $T$  arithmetic operations, Protocol 8 has a complexity of  $2T$  plus  $4n$  additions and  $4n$  multiplications. The value of  $T$  depends on the locality  $d$  and the exact choice of the predicate that computes  $G_i$ , and deserves further study. Recall that Protocol 8 essentially realizes “pseudorandom” batch-OLE. This can be upgraded to a “standard” batch-OLE with an additional overhead of  $2n$  multiplications and  $5n$  additions.

**Acknowledgement.** We are grateful to Ivan Damgård and Yuval Ishai for early discussions that influenced this work. We also thank YI for explaining various aspects of [17, 18]. We thank the reviewers of Eurocrypt2023 for their comments.

## References

1. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: how to sell digital goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (May 2001). [https://doi.org/10.1007/3-540-44987-6\\_8](https://doi.org/10.1007/3-540-44987-6_8)
2. Alekhnovich, M.: More on average case vs approximation complexity. In: 44th FOCS, pp. 298–307. IEEE Computer Society Press (October 2003). <https://doi.org/10.1109/SFCS.2003.1238204>
3. Applebaum, B.: Cryptographic hardness of random local functions. *Comput. Complex.* **25**(3), 667–722 (2015). <https://doi.org/10.1007/s00037-015-0121-8>
4. Applebaum, B., Avron, J., Brzuska, C.: Arithmetic cryptography. *J. ACM* **64**(2), 10:1–10:74 (2017). <https://doi.org/10.1145/3046675>
5. Applebaum, B., Damgård, I., Ishai, Y., Nielsen, M., Zichron, L.: Secure arithmetic computation with constant computational overhead. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 223–254. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63688-7\\_8](https://doi.org/10.1007/978-3-319-63688-7_8)
6. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in  $nc^0$ . *SIAM J. Comput.* **36**(4), 845–888 (2006). <https://doi.org/10.1137/S0097539705446950>
7. Applebaum, B., Ishai, Y., Kushilevitz, E.: On pseudorandom generators with linear stretch in  $nc^0$ . *Comput. Complex.* **17**(1), 38–69 (2008). <https://doi.org/10.1007/s00037-007-0237-6>
8. Applebaum, B., Ishai, Y., Kushilevitz, E.: How to garble arithmetic circuits. *SIAM J. Comput.* **43**(2), 905–929 (2014). <https://doi.org/10.1137/120875193>
9. Applebaum, B., Kachlon, E.: Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. In: Zuckerman, D. (ed.) 60th FOCS. pp. 171–179. IEEE Computer Society Press (November 2019). <https://doi.org/10.1109/FOCS.2019.00020>
10. Applebaum, B., Konstantini, N.: Actively secure arithmetic computation and vole with constant computational overhead. *Cryptology ePrint Archive*, Paper 2023/270 (2023). <https://eprint.iacr.org/2023/270>, <https://eprint.iacr.org/2023/270>
11. Applebaum, B., Lovett, S.: Algebraic attacks against random local functions and their countermeasures. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 1087–1100. ACM Press (June 2016). <https://doi.org/10.1145/2897518.2897554>



12. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer extensions with security for malicious adversaries. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 673–701. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46800-5\\_26](https://doi.org/10.1007/978-3-662-46800-5_26)
13. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac’n’Cheese: zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 92–122. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84259-8\\_4](https://doi.org/10.1007/978-3-030-84259-8_4)
14. Beaver, D.: Correlated pseudorandomness and the complexity of private computations. In: 28th ACM STOC. pp. 479–488. ACM Press (May 1996). <https://doi.org/10.1145/237814.237996>
15. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10626, pp. 336–365. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-70700-6\\_12](https://doi.org/10.1007/978-3-319-70700-6_12)
16. Bordewijk, J.L.: Inter-reciprocity applied to electrical networks. *Appl. Sci. Res. Sect. A* **6**(1), 1–74 (1957)
17. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018, pp. 896–912. ACM Press (October 2018). <https://doi.org/10.1145/3243734.3243868>
18. Boyle, E., et al.: Efficient two-round OT extension and silent non-interactive secure computation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 291–308. ACM Press (November 2019). <https://doi.org/10.1145/3319535.3354255>
19. Chase, M., et al.: Reusable non-interactive secure computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 462–488. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-26954-8\\_15](https://doi.org/10.1007/978-3-030-26954-8_15)
20. Chor, B., Goldreich, O., Håstad, J., Friedman, J., Rudich, S., Smolensky, R.: The bit extraction problem of  $t$ -resilient functions (preliminary version). In: 26th FOCS. pp. 396–407. IEEE Computer Society Press (October 1985). <https://doi.org/10.1109/SFCS.1985.55>
21. Crépeau, C.: Equivalence between two Flavours of oblivious transfers. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 350–354. Springer, Heidelberg (1988). [https://doi.org/10.1007/3-540-48184-2\\_30](https://doi.org/10.1007/3-540-48184-2_30)
22. Crépeau, C., Kilian, J.: Weakening security assumptions and oblivious transfer. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 2–7. Springer, New York (1990). [https://doi.org/10.1007/0-387-34799-2\\_1](https://doi.org/10.1007/0-387-34799-2_1)
23. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-point zero knowledge and its applications. In: Tessaro, S. (ed.) ITC 2021. LIPIcs, vol. 199, pp. 5:1–5:24. Schloss Dagstuhl (2021). <https://doi.org/10.4230/LIPIcs.ITC.2021.5>
24. Druk, E.: Linear time encodable codes and cryptography. Master’s thesis, Technion (2013)
25. Druk, E., Ishai, Y.: Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In: Naor, M. (ed.) ITCS 2014. pp. 169–182. ACM (January 2014). <https://doi.org/10.1145/2554797.2554815>
26. 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](https://doi.org/10.1007/978-3-540-30576-7_17)
27. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: Shmoys, D.B. (ed.)

- 46th ACM STOC, pp. 495–504. ACM Press (May/June 2014). <https://doi.org/10.1145/2591796.2591861>
28. Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting data privacy in private information retrieval schemes. In: 30th ACM STOC, pp. 151–160. ACM Press (May 1998). <https://doi.org/10.1145/276698.276723>
  29. Ghosh, S., Nielsen, J.B., Nilges, T.: Maliciously secure oblivious linear function evaluation with constant overhead. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 629–659. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-70694-8\\_22](https://doi.org/10.1007/978-3-319-70694-8_22)
  30. 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](https://doi.org/10.1007/978-3-030-17659-4_6)
  31. Gilboa, N.: Two party RSA key generation. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 116–129. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48405-1\\_8](https://doi.org/10.1007/3-540-48405-1_8)
  32. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987). <https://doi.org/10.1145/28395.28420>
  33. Guruswami, V., Indyk, P.: Linear-time encodable/decodable codes with near-optimal rate. *IEEE Trans. Inf. Theory* **51**(10), 3393–3400 (2005). <https://doi.org/10.1109/TIT.2005.855587>
  34. 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](https://doi.org/10.1007/978-3-540-45146-4_9)
  35. Ishai, Y., Kushilevitz, E.: Private simultaneous messages protocols with applications. In: Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, 17–19 June 1997, Proceedings, pp. 174–184. IEEE Computer Society (1997). <https://doi.org/10.1109/ISTCS.1997.595170>
  36. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: a new representation with applications to round-efficient secure computation. In: 41st FOCS, pp. 294–304. IEEE Computer Society Press (November 2000). <https://doi.org/10.1109/SFCS.2000.892118>
  37. Ishai, Y., Kushilevitz, E.: Perfect constant-round secure computation via perfect randomizing polynomials. In: Widmayer, P., et al. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 244–256. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-45465-9\\_22](https://doi.org/10.1007/3-540-45465-9_22)
  38. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 433–442. ACM Press (May 2008). <https://doi.org/10.1145/1374376.1374438>
  39. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer – efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-85174-5\\_32](https://doi.org/10.1007/978-3-540-85174-5_32)
  40. Ishai, Y., Prabhakaran, M., Sahai, A.: Secure arithmetic computation with no honest majority. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 294–314. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-00457-5\\_18](https://doi.org/10.1007/978-3-642-00457-5_18)
  41. Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically secure protocols and security under composition. *SIAM J. Comput.* **39**(5), 2090–2112 (2010). <https://doi.org/10.1137/090755886>, <https://doi.org/10.1137/090755886>
  42. Lindell, Y.: General composition and universal composability in secure multi-party computation. In: 44th FOCS, pp. 394–403. IEEE Computer Society Press (October 2003). <https://doi.org/10.1109/SFCS.2003.1238213>

43. Mohassel, P., Weinreb, E.: Efficient secure linear algebra in the presence of covert or computationally unbounded adversaries. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 481–496. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-85174-5\\_27](https://doi.org/10.1007/978-3-540-85174-5_27)
44. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: 31st ACM STOC, pp. 245–254. ACM Press (May 1999). <https://doi.org/10.1145/301250.301312>
45. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Kosaraju, S.R. (ed.) 12th SODA, pp. 448–457. ACM-SIAM (January 2001)
46. 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](https://doi.org/10.1007/978-3-540-85174-5_31)
47. Rabin, M.O.: How to exchange secrets with oblivious transfer. IACR Cryptol. ePrint Arch. p. 187 (2005), <http://eprint.iacr.org/2005/187>
48. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-OLE: Improved constructions and implementation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 1055–1072. ACM Press (November 2019). <https://doi.org/10.1145/3319535.3363228>
49. Shankar, B., Srinathan, K., Rangan, C.P.: Alternative protocols for generalized oblivious transfer. In: Rao, S., Chatterjee, M., Jayanti, P., Murthy, C.S.R., Saha, S.K. (eds.) ICDCN 2008. LNCS, vol. 4904, pp. 304–309. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-77444-0\\_31](https://doi.org/10.1007/978-3-540-77444-0_31)
50. Tassa, T.: Generalized oblivious transfer by secret sharing. Des. Codes Cryptogr. **58**(1), 11–21 (2011). <https://doi.org/10.1007/s10623-010-9378-8>
51. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24–27 May 2021, pp. 1074–1091. IEEE (2021). <https://doi.org/10.1109/SP40001.2021.00056>
52. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press (October 1986). <https://doi.org/10.1109/SFCS.1986.25>
53. Zichron, L.: Locally computable arithmetic pseudorandom generators. Master’s thesis, Tel Aviv University (2017), available from Applebaum’s home page



# SUPERPACK: Dishonest Majority MPC with Constant Online Communication

Daniel Escudero<sup>1</sup>, Vipul Goyal<sup>2,3</sup>, Antigoni Polychroniadou<sup>1</sup>, Yifan Song<sup>4</sup>,  
and Chenkai Weng<sup>5</sup>

<sup>1</sup> J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE, New York, NY, USA  
{daniel.escudero,antigoni.polychroniadou}@jpmorgan.com

<sup>2</sup> NTT Research, Palo Alto, CA, USA

<sup>3</sup> Carnegie Mellon University, Pittsburgh, PA, USA  
vipul@cmu.edu

<sup>4</sup> Tsinghua University, Beijing, China  
yfsong1995@gmail.com

<sup>5</sup> Northwestern University, Evanston, IL, USA  
ckweng@u.northwestern.edu

**Abstract.** In this work we present a novel actively secure dishonest majority MPC protocol, SUPERPACK, whose efficiency improves as the number of *honest* parties increases. Concretely, let  $0 < \epsilon < 1/2$  and consider an adversary that corrupts  $t < n(1 - \epsilon)$  out of  $n$  parties. SUPERPACK requires  $6/\epsilon$  field elements of online communication per multiplication gate across all parties, assuming circuit-dependent preprocessing, and  $10/\epsilon$  assuming circuit-independent preprocessing. In contrast, most of previous works such as SPDZ (Damgård *et al.*, ESORICS 2013) and its derivatives perform the same regardless of whether there is only one honest party, or a constant (non-majority) fraction of honest parties. The only exception is due to Goyal *et al.* (CRYPTO 2022), which achieves  $58/\epsilon + 96/\epsilon^2$  field elements assuming circuit-independent preprocessing. Our work improves this result substantially by a factor of at least 25 in the circuit-independent preprocessing model.

Practically, we also compare our work with the best concretely efficient online protocol Turbospeedz (Ben-Efraim *et al.*, ACNS 2019), which achieves  $2(1 - \epsilon)n$  field elements per multiplication gate among all parties. Our online protocol improves over Turbospeedz as  $n$  grows, and as  $\epsilon$  approaches  $1/2$ . For example, if there are 90% corruptions ( $\epsilon = 0.1$ ), with  $n = 50$  our online protocol is  $1.5\times$  better than Turbospeedz and with  $n = 100$  this factor is  $3\times$ , but for 70% corruptions ( $\epsilon = 0.3$ ) with  $n = 50$  our online protocol is  $3.5\times$  better, and for  $n = 100$  this factor is  $7\times$ .

Our circuit-dependent preprocessing can be instantiated from OLE/VOLE. The amount of OLE/VOLE correlations required in our work is a factor of  $\approx \epsilon n/2$  smaller than these required by Le Mans (Rachuri and Scholl, CRYPTO 2022) leveraged to instantiate the preprocessing of Turbospeedz.

Our dishonest majority protocol relies on packed secret-sharing and leverages ideas from the honest majority TURBOPACK (Escudero *et al.*,

CCS 2022) protocol to achieve concrete efficiency for any circuit topology, not only SIMD. We implement both SUPERPACK and Turbospeedz and verify with experimental results that our approach indeed leads to more competitive runtimes in distributed environments with a moderately large number of parties.

## 1 Introduction

Secure multiparty computation (MPC) protocols enable a set of parties  $P_1, \dots, P_n$  to securely compute a function on their private inputs while leaking only the final output. MPC protocols remain secure even if  $t$  out of the  $n$  parties are corrupted. There are honest majority protocols, which are designed to tolerate at most a minority of corruptions, or in other words, they assume that  $t < n/2$ . On the other hand, protocols in the dishonest majority setting accommodate  $t \geq n/2$ . Honest majority MPC protocols can offer information-theoretic security (that is, they do not need to depend on computational assumptions, which also makes them more efficient), or guaranteed output delivery (that is, all honest parties are guaranteed to receive the output of the computation). However, dishonest majority protocols tolerate a larger number of corruptions at the expense of relying on computational assumptions and sacrificing fairness and guarantee output delivery.

Communication complexity is a key measure of efficiency for MPC. Over the last few decades, great progress has been made in the design of communication-efficient honest majority protocols [4, 7, 9, 13, 15, 18, 20, 23, 24]. In particular, the recent work [15] shows that it is possible to achieve *constant* communication complexity among all parties (i.e.,  $O(1)$ ) per multiplication gate in the online phase while maintaining *linear* communication complexity in the number of parties (i.e.,  $O(n)$ ) per multiplication gate in the offline phase — which is independent of the private inputs.

Dishonest majority protocols provide the best security guarantees in terms of collusion sizes since security will be ensured even if all parties but one jointly collude against the remaining honest party. It is known that in this setting public key cryptography tools are needed. In the seminar work of Beaver [1] it was shown how to push most of the “heavy crypto machinery” to an offline phase, hence allowing for a more efficient online phase that can even be information-theoretically secure, or at least use simpler cryptographic tools such as PRGs and hash functions for efficiency. This approach eventually led to the seminal works of BeDOZa [5] and SPDZ [12, 14], which leveraged the Beaver triple technique from [1] together with message authentication codes to achieve a concretely efficient online phase with linear linear communication complexity in the number of parties per gate. The online phase in SPDZ has been very influential, and there is a large body of research that has focused solely on improving the offline phase, leaving the SPDZ online phase almost intact.

Despite the progress of designing MPC in the dishonest majority setting, it remains unclear whether we can achieve a sub-linear communication complexity

in the number of parties per multiplication gate without substantially sacrificing the offline phase<sup>1</sup>. This motivates us to study the following question:

*“If a small constant fraction of parties are honest, can we build concretely efficient dishonest majority MPC protocols that achieve constant online communication among all parties per multiplication gate with comparable efficiency as the state-of-the-art in the honest majority setting?”*

To be concretely efficient, we refer to protocols that do not rely on heavy Cryptographic tools such as FHE. In particular, we restrict the online phase to be almost information-theoretic except the black-box use of PRGs or hash functions. Perhaps surprisingly, it is not clear what benefits can be achieved if assume instead of all-but-one corruption, but a constant fraction of parties are honest. In fact, in the case that there are  $n - t > 1$  honest parties — unless these constitute a majority — the best one can do to optimize communication is removing  $(n - t - 1)$  parties so that, in the new set, there is at least *one* honest party, which is the only requirement for dishonest majority protocols to guarantee security. To the best of our knowledge, the only exception to this is [22], which considers the corruption threshold  $t = n(1 - \epsilon)$  for a constant  $\epsilon$  in the circuit-independent preprocessing model and achieves  $58/\epsilon + 96/\epsilon^2$  elements per multiplication gate among all parties in the malicious security setting<sup>2</sup>. Despite the constant communication complexity per multiplication gate achieved in [22], it requires hundreds or even thousands of parties to outperform SPDZ [14].

Given the above state-of-affairs, we see that existing dishonest majority protocols are either not very flexible in terms of the amount of corruptions — 50% corruptions are as good as 99%, and having more honest parties do not provide any substantial benefit — or not concretely efficient at all.

## 1.1 Our Contribution

In this work, we answer the above question affirmatively: we design the first concretely efficient dishonest majority MPC protocol SUPERPACK that achieves constant online communication among all parties per multiplication gate with comparable efficiency as the state-of-the-art in the honest majority setting [15]. SuperPack tolerates any number of corruptions and becomes more efficient as the number of honest parties increases, or put differently, it becomes more efficient as the percentage of corrupted parties decreases.

More concretely, we show the following theorem.

**Theorem 1 (Informal).** *Let  $n$  be a positive integer,  $\epsilon \in (0, 1/2)$  be a constant, and  $\kappa$  be the security parameter. For an arithmetic circuit  $C$  that computes an  $n$ -ary functionality  $\mathcal{F}$ , there exists an  $n$ -party protocol that computes  $C$*

<sup>1</sup> An example is [10] which achieves slightly sub-linear communication complexity in the *circuit size* at the cost of increasing the preprocessed data size to be quadratic in the circuit size.

<sup>2</sup> The work [22] does not analyze the concrete cost of their malicious protocol. We obtain this number by counting the amount of communication in their construction. We note that the protocol in [22] also needs to interact for addition gates. Our reported number assumes that the amount of addition gates is the same as the amount of multiplication gates.

with computational security against a fully malicious adversary who can control at most  $t = n(1 - \epsilon)$  corrupted parties. The protocol has total communication  $O(6|C|n + 45|C|/\epsilon)$  elements (ignoring the terms that are independent of the circuit size or only related to the circuit depth<sup>3</sup>) with splitting cost:

- *Online Phase*:  $6/\epsilon$  per multiplication gate across all parties.
- *Circuit-Dependent Preprocessing Phase*:  $4/\epsilon$  per multiplication gate across all parties.
- *Circuit-Independent Preprocessing Phase*:  $6n + 35/\epsilon$  per multiplication gate across all parties.

Our construction has the following features:

**Online phase (Section 4).** The online phase requires *circuit-dependent* preprocessing (meaning, this preprocessing does not depend on the inputs but it depends on the topology of the underlying circuit). It relies on information-theoretic tools and as it is typical we also introduce PRGs to further improve the efficiency.

**Circuit-dependent offline phase (Section 5).** The circuit-dependent preprocessing is instantiated using *circuit-independent* preprocessing (meaning, it may depend on the amount of certain types of gates of the circuit, but not on its topology) in a simple and efficient manner. Again, the protocol makes use of information-theoretical tools together with PRGs to further improve the efficiency.

**Circuit-independent offline phase (Section 6).** The circuit-independent preprocessing is instantiated by a vector oblivious linear evaluation (VOLE) functionality and an oblivious linear evaluation (OLE) functionality. These two functionalities are realized by protocols in Le Mans [25], which can achieve sub-linear communication complexity in the amount of preprocessed data. In addition, we manage to reduce the amount of preprocessed data by a factor of  $\epsilon n/2$  compared with that in [25]. More discussion can be found in Sect. 2.

*Comparison to Best Previous Works.* When comparing with [22], which achieves  $58/\epsilon + 96/\epsilon^2$  elements per multiplication gate among all parties in the circuit-independent preprocessing phase, our protocol achieves a factor of at least 25 improvement in the same setting, and a factor of at least 40 improvement in the circuit-dependent preprocessing phase. Since [22] does not realize the circuit-independent preprocessing phase, we do not compare the cost in the circuit-independent preprocessing phase.

Since our goal is to optimize the online phase of dishonest majority protocols where there is a constant fraction of honest parties, we take as a baseline for comparison the existing dishonest majority protocol with the best concrete

---

<sup>3</sup> The only term that is related to the circuit depth is in the form of  $O(n \cdot \text{Depth})$ . This is because of the use of packed secret sharing which requires to evaluate at least  $O(n)$  gates per layer. A similar term also occurs in previous works that use packed secret sharings [2, 11, 15, 17, 21, 22].

efficiency in the online phase. This corresponds to the Turbospeedz protocol [3], which is set in the circuit-dependent preprocessing model. To instantiate the preprocessing, we utilize the state-of-the-art [25]. Details on this protocol are given in the full version of this paper. The resulting protocol has the following communication complexity:  $2(1 - \epsilon)n$  in the online phase,  $4(1 - \epsilon)n$  in the circuit-dependent offline phase, and  $6(1 - \epsilon)n$  in the circuit-independent offline phase when instantiated using Le Mans [25] (ignoring the calls to the VOLE and OLE functionalities). Again, the VOLE and OLE functionalities can be properly instantiated with sub-linear communication complexity in the preprocessed data size. And our protocol even reduce this size by a factor of  $\epsilon n/2$ .

The communication complexity of our protocol and its comparison with respect to Turbospeedz is given in Table 1. We see that our online phase is better than Turbospeedz by a factor of  $(n\epsilon(1 - \epsilon))/3$ . Some observations about this expression:

- (*Fixing the ratio  $\epsilon$* ). Given a factor  $\epsilon$ , meaning there is an  $\epsilon \times 100\%$  percentage of honest parties and  $(1 - \epsilon) \times 100\%$  percentage of corrupt parties, our online phase is better as long as the total number of parties  $n$  is at least the *constant* term  $3/(\epsilon(1 - \epsilon))$ , with the improvement factor increasing as  $n$  increases past this threshold. Furthermore, this term goes down as  $\epsilon$  approaches  $1/2$ , meaning that the more honest parties/less corruptions, the smaller  $n$  needs to be for our online phase to be better. For example, if  $\epsilon = 0.1$  (90% corruptions) we see improvements with  $n \geq 34$ ; if  $\epsilon = 0.2$  (80% corruptions) then  $n \geq 19$ ; and if  $\epsilon = 0.3$  (70% corruptions) then  $n \geq 15$ .
- (*Fixing the number of honest parties*). Given a *fixed* number of *honest* parties  $h$ , our online protocol is  $(\frac{h}{4}) \times$  better than prior work *regardless of the total number of parties*  $n$ , as long as  $n \geq 4h$ . This is proven in the full version of this paper. This motivates the use of our protocol over prior solutions for any number of parties, as long as a minimal support of honest parties can be assumed.

Regarding the complete offline phase (ignoring calls to  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  and  $\mathcal{F}_{\text{nVOLE}}$ ), our complexity is  $6n + 39/\epsilon$ , while in Turbospeedz it is  $10(1 - \epsilon)n$ . In the limit as  $n \rightarrow \infty$ , our offline protocol is approximately a factor of  $10(1 - \epsilon)/6$  times better than Turbospeedz/Le Mans, which ranges between  $10/6 \approx 1.6$  for  $\epsilon = 0$ , to  $5/6 \approx 0.83$  for  $\epsilon = 1/2$ . As a result, *in the limit*, our offline phase is only  $1/0.83 = 1.2 \times$  less efficient than that of Turbospeedz (and for  $\epsilon$  close to zero it can be even up to 1.6 better), which is a reasonable cost taking into account the benefits in the online phase. A more thorough discussion on the communication complexity and its implications is given in the full version of this paper.

*Implementation and Experimental Results.* Finally, we implement all of our protocol—except for the OLE/VOLE functionalities—and verify that, experimentally, our protocol outperforms Turbospeedz by the expected amount based on the communication measures when the runtimes are not computation bound. For example, in a 100 mbps network our online phase is more than  $\approx 4.5 \times$  better than that of Turbospeedz for 80 parties, where 60% of them are corrupted. If



**Table 1.** Communication complexity in terms of field elements per multiplication gate of SUPERPACK, and comparison to the previous work with the best concrete efficiency in the online phase, which is Turbospeedz [3] (with its offline phase instantiated by Le Mans [25]), referred to as Turbospeedz\*. The cost of the calls to  $\mathcal{F}_{\text{OLE}}^{\text{PROG}}$  and  $\mathcal{F}_{\text{NVOLE}}$  in the circuit-independent offline phase is ignored.

	Online	CD Offline	CI Offline
SuperPack	$6/\epsilon$	$4/\epsilon$	$6n + 35/\epsilon$
Turbospeedz*	$2(1 - \epsilon)n$	$4(1 - \epsilon)n$	$6(1 - \epsilon)n$

the network is too fast, then computation becomes a more noticeable bottleneck, and our improvements are less noticeable. This is discussed in Sect. 7.

## 2 Overview of the Techniques

In this section we provide an overview of our SUPERPACK protocol. Recall that in our setting we have  $t < n(1 - \epsilon)$ . Let  $\mathbb{F}$  be a finite field with  $|\mathbb{F}| \geq 2^\kappa$ , where  $\kappa$  is the security parameter. We consider packed Shamir secret sharing, where  $k$  secrets  $\mathbf{x} = (x_1, \dots, x_k)$  are turned into shares as  $[\mathbf{x}]_d = (f(1), \dots, f(n))$ , where  $f(\mathbf{x})$  is a uniformly random polynomial over  $\mathbb{F}$  of degree at most  $d$  constrained to  $f(0) = x_1, \dots, f(-(k - 1)) = x_k$ . It also holds that  $[\mathbf{x}]_{d_1} * [\mathbf{y}]_{d_2} = [\mathbf{x} * \mathbf{y}]_{d_1 + d_2}$ , where the operator  $*$  denotes point-wise multiplication. In our protocol we would like to be able to multiply degree- $d$  sharings by degree- $(k - 1)$  sharings (which corresponds to multiplying by constants), so we would like the sum of these degrees to be at most  $n - 1$  so that the  $n$  parties determine the underlying secrets. For this, we take  $d + (k - 1) = n - 1$ . On the other hand, we also want the secrets of a degree- $d$  packed Shamir sharing to be private against  $t$  corrupted parties, which requires  $d \geq t + k - 1$ . Together, these imply  $n = t + 2(k - 1) + 1 = t + 2k - 1$ , and  $k = \frac{n - t + 1}{2} \geq \frac{\epsilon n + 2}{2}$ .

At a high level, our technical contributions can be summarized as two aspects:

1. First, we lift the online protocol of TURBOPACK [15] from the honest majority setting to the dishonest majority setting. Our starting point is the observation that the passive version of the online protocol from TURBOPACK [15] also works for a dishonest majority by setting the parameters correctly. To achieve malicious security, however, the original techniques do not work. This is because in TURBOPACK, all parties will prepare a degree- $t$  Shamir sharing for each wire value in the circuit. In the honest majority setting, a degree- $t$  Shamir sharing satisfies that the shares of honest parties can fully determine the secret, and the most that malicious parties can do is to change their local shares and cause the whole sharing inconsistent (in the sense that the shares do not lie on a degree- $t$  polynomial). Malicious parties however cannot change the secret by changing their shares. This property unfortunately does not hold in the dishonest majority setting.

Instead, in our case, we rely on a different type of redundancy widely used in the dishonest majority setting: We make use of message authentication codes, or MACs, to ensure that corrupted parties cannot change the secrets by changing their local shares without being caught. While a similar technique has also been used in [22], their way of using MACs increases the online communication complexity by a factor of at least 2 compared with their passive protocol.

We will show how to use MACs in a way such that the online communication complexity remains the same as our passive protocol.

2. Second, we have to reinvent the circuit-independent preprocessing protocol for SUPERPACK as the corresponding protocol from TURBOPACK highly relies on the assumption of honest majority, plus that we also need the preprocessed sharings to be authenticated due to the larger corruption threshold.

The main preprocessing data we need to prepare is referred to as *Packed Beaver Triples*, which are first introduced in [22]. At a high level, a packed Beaver triple contains three packed Shamir sharings ( $[a], [b], [c]$ ) such that  $a, b$  are random vectors in  $\mathbb{F}^k$  and  $c = a * b$ . To prepare such a packed Beaver triple, a direct approach would be first preparing standard Beaver triples using additive sharings and then transform them to packed Shamir sharings. In this way, we may reuse the previous work of generating standard Beaver triples in a black box way. However, this idea requires us to not only pay the cost of preparing standard Beaver triples, but also pay the cost of doing the sharing transformation. The direct consequence is that the overall efficiency of our protocol will be *worse* than that of the state-of-the-art [25] in the dishonest majority setting. (And this is the approach used in TURBOPACK [15].)

We will show how to take the advantage of the constant fraction of honest parties in the circuit-independent preprocessing phase by carefully using the techniques of [25] in our setting.

In the following, we will start with a sketch of the modified passive version of TURBOPACK, which is suitable in our setting.

## 2.1 Starting Point: TURBOPACK

Our starting point is the observation that the passive version of the online protocol from TURBOPACK [15], which is set in the *honest majority* setting, also works for a dishonest majority by setting the parameters correctly. We focus mostly on multiplication gates. So we ignore details regarding input and output gates.

**Preprocessing.** We consider an arithmetic circuit whose wires are indexed by certain identifiers, which we denote using lowercase Greek letters  $\alpha, \beta, \gamma$ , etc. Our work is set in the client-server model where there are input and output gates associated to *clients*, who will be in charge of providing input/receiving output. Each multiplication layer of the circuit is split into *batches* of size  $k$ . Similarly, each input and output layer assigned to a given client are split into batches of

size  $k$ . The invariant in TURBOPACK is the following. First, every wire  $\alpha$  that is not the output of an addition gate has associated to it a uniformly random value  $\lambda_\alpha$ . If a wire  $\gamma$  is the output of an addition gate with input wires  $\alpha$  with wire  $\beta$ , then  $\lambda_\gamma$  is defined (recursively) as  $\lambda_\alpha + \lambda_\beta$ .

The parties are assumed to have the following (circuit-dependent) preprocessing material: For every group of  $k$  multiplication gates with input wires  $\alpha, \beta$  and output wires  $\gamma$ , the parties have  $[\lambda_\alpha]_{n-k}$ ,  $[\lambda_\beta]_{n-k}$ , and  $[\lambda_\gamma]_{n-1}$  (The degree of the last sharing is chosen to be  $n-1$  on purpose). In addition, all parties also hold a fresh packed Beaver triple  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, [\mathbf{c}]_{n-1})$  for this gate (Again, the degree of the last sharing is chosen to be degree- $(n-1)$  on purpose).

**Main Invariant.** The main invariant in TURBOPACK is that for every wire  $\alpha$ ,  $P_1$  knows the value  $\mu_\alpha = v_\alpha - \lambda_\alpha$ , where  $v_\alpha$  denotes the actual value in wire  $\alpha$  for a given choice of inputs. Notice that this invariant preserves the privacy of all intermediate wires, since  $P_1$  only learns a masked version of the wire values, and the masks, the  $\lambda_\alpha$ 's, are uniformly random and they are kept private with packed Shamir sharings of degree  $n-k = t + (k-1)$ . We now discuss how, in the original TURBOPACK work, this invariant is maintained throughout the circuit execution. We only focus on (groups of) multiplication gates. Addition gates can be processed locally. Groups of *input* gates with wires  $\alpha$  make use of a simple protocol in which the client who owns the gates learns the corresponding masks  $\lambda_\alpha$ , and sends  $\mu_\alpha = v_\alpha - \lambda_\alpha$  to  $P_1$ . Groups of output gates are handled in a similar way.

*Maintaining the Invariant for Multiplication Gates.* Consider a group of multiplication gates in a given circuit level, having input wires  $\alpha, \beta$ , and output wires  $\gamma$ . Assume that the invariant holds for the input wires, meaning that  $P_1$  knows  $\mu_\alpha = v_\alpha - \lambda_\alpha$  and  $\mu_\beta = v_\beta - \lambda_\beta$ . Recall that the parties have the preprocessed sharings  $[\lambda_\alpha]_{n-k}$ ,  $[\lambda_\beta]_{n-k}$ , and  $[\lambda_\gamma]_{n-1}$ . To maintain the invariant,  $P_1$  must learn  $\mu_\gamma = v_\gamma - \lambda_\gamma$ , where  $v_\gamma = v_\alpha * v_\beta$ . This is achieved by using the techniques of packed Beaver triples introduced in [22]. Recall that all parties also hold a fresh packed Beaver triple  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, [\mathbf{c}]_{n-1})$ . All parties proceeds as follows:

1. All parties locally compute the packed Shamir sharing  $[\lambda_\alpha - \mathbf{a}]_{n-k} = [\lambda_\alpha]_{n-k} - [\mathbf{a}]_{n-k}$  and let  $P_1$  learn  $\lambda_\alpha - \mathbf{a}$ . Similar step is done to let  $P_1$  learn  $\lambda_\beta - \mathbf{b}$ .
2.  $P_1$  computes  $v_\alpha - \mathbf{a} = \mu_\alpha + (\lambda_\alpha - \mathbf{a})$  and computes  $v_\beta - \mathbf{b}$  similarly. Then,  $P_1$  distributes shares  $[v_\alpha - \mathbf{a}]_{k-1}$  and  $[v_\beta - \mathbf{b}]_{k-1}$  to the parties.
3. Using the received shares and the shares obtained in the preprocessing phase, the parties compute locally

$$\begin{aligned} [v_\gamma]_{n-1} &= [v_\alpha - \mathbf{a}]_{k-1} * [v_\beta - \mathbf{b}]_{k-1} + [v_\alpha - \mathbf{a}]_{k-1} * [\mathbf{b}]_{n-k} \\ &\quad + [v_\beta - \mathbf{b}]_{k-1} * [\mathbf{a}]_{n-k} + [\mathbf{c}]_{n-1}. \end{aligned}$$

and  $[\mu_\gamma]_{n-1} = [v_\gamma]_{n-1} - [\lambda_\gamma]_{n-1}$ .

4. The parties send their shares  $[\mu_\gamma]_{n-1}$  to  $P_1$ , who reconstructs  $\mu_\gamma$ . It is easy to see that  $\mu_\gamma = \mathbf{v}_\alpha * \mathbf{v}_\beta - \lambda_\gamma$ .

Note that the first step can be completely moved to the circuit-dependent preprocessing phase since both  $[\lambda_\alpha]_{n-k}$  and  $[\mathbf{a}]_{n-k}$  are preprocessed data. With this optimization, the online protocol only requires all parties to communicate  $3n$  elements for  $k = \epsilon n/2$  multiplication gates, which is  $6/\epsilon$  elements per gate among all parties.

## 2.2 Achieving Active Security

There are multiple places where an active adversary can cheat in the previous protocol, with the most obvious being distributing incorrect (or even invalid)  $[\mathbf{v}_\alpha - \mathbf{a}]_{k-1}$  and  $[\mathbf{v}_\beta - \mathbf{b}]_{k-1}$  at a group of multiplication gates, either by corrupting  $P_1$ , or by sending incorrect shares in previous gates to  $P_1$ . This is prevented in TURBOPACK by explicitly making use of the honest majority assumption: Using the degree- $(k-1)$  packed Shamir sharings distributed by  $P_1$ , the parties will be able to obtain a certain “individual” (*i.e.* non-packed) degree- $t$  Shamir sharing for each wire value. As we discussed above, a degree- $t$  Shamir sharing in the honest majority setting allows honest parties to fully determine the secret. This enables the use of distributed zero-knowledge techniques [6] to check the correctness of the computation.

In our case where  $t \geq n/2$ , these techniques cannot be used. Instead, we rely on a different type of redundancy widely used in the dishonest majority setting, namely, we make use of message authentication codes, or MACs, to ensure the parties cannot deviate from the protocol execution when performing actions like reconstructing secret-shared values. We observe that the use of MACs has the following two advantages:

- With MACs, corrupted parties cannot change the secrets of a degree- $(n-k)$  packed Shamir sharing without being detected except with a negligible probability.
- In addition to adding verifiability to packed Shamir sharings, we show how to allow all parties to directly compute MACs of the secret values that are shared by  $P_1$  using degree- $(k-1)$  packed Shamir sharings. This allows us to directly verify whether  $\mathbf{v}_\alpha - \mathbf{a}$  and  $\mathbf{v}_\beta - \mathbf{b}$  are correct without doing distributed zero-knowledge like [15].

Before we describe our approach, let us introduce some notation. We use  $[x]_i_t$  to denote a Shamir secret sharing of degree  $t$ , where the secret is in position  $-(i-1)$ . *I.e.*, the corresponding polynomial  $f(\mathbf{x})$  satisfies that  $f(-(i-1)) = x$ . We also use  $\langle x \rangle$  to denote an additive secret sharing of  $x$ . Observe that from a Shamir sharing of  $x$  (or a packed Shamir sharing that contains  $x$ ), all parties can locally obtain an additive sharing of  $x$  by locally multiplying suitable Lagrange coefficients.

To achieve active security, we need the parties to hold preprocessing data of the following form:

- Shares of a global random key  $\Delta \in \mathbb{F}$  in the form  $([\Delta|_1]_t, \dots, [\Delta|_k]_t)$ .
- For every group of  $k$  multiplication gates with input wires  $\alpha, \beta$  and output wires  $\gamma$ , recall that all parties hold a fresh packed Beaver triple  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, [\mathbf{c}]_{n-1})$ . They additionally hold  $[\Delta \cdot \mathbf{a}]_{n-k}$ ,  $[\Delta \cdot \mathbf{b}]_{n-k}$ , and  $\{\langle \Delta \cdot c_i \rangle\}_{i=1}^k$ , and also  $\{\langle \Delta \cdot \lambda_{\gamma_i} \rangle\}_{i=1}^k$ .

With these at hand, the new invariant we maintain to ensure active security is that (1) as before,  $P_1$  learns  $\mu_\alpha$  and  $\lambda_\alpha - \mathbf{a}$  for every group of input wires  $\alpha$  of multiplication gates, but in addition (2) the parties have shares  $\langle \Delta \cdot \mu_{\alpha_i} \rangle$  and  $\langle \Delta \cdot (\lambda_{\alpha_i} - a_i) \rangle$  for all  $i \in \{1, \dots, k\}$ . In this way, the first part of the invariant enables the parties to compute the circuit, while the second ensures that  $P_1$  distributed correct values.

*Maintaining the New Invariant.* Consider a group of multiplication gates with input wires  $\alpha, \beta$ , and output wires  $\gamma$ . Assume that the invariant holds for the input wires, meaning that  $P_1$  knows  $\mu_\alpha = \mathbf{v}_\alpha - \lambda_\alpha$  and  $\mu_\beta = \mathbf{v}_\beta - \lambda_\beta$  as well as  $\lambda_\alpha - \mathbf{a}$  and  $\lambda_\beta - \mathbf{b}$ , and also the parties have  $\{(\langle \Delta \cdot \mu_{\alpha_i} \rangle, \langle \Delta \cdot (\lambda_{\alpha_i} - a_i) \rangle)\}_{i=1}^k$  and  $\{(\langle \Delta \cdot \mu_{\beta_i} \rangle, \langle \Delta \cdot (\lambda_{\beta_i} - b_i) \rangle)\}_{i=1}^k$ .

The parties preserve the invariant as follows.

- For (1), we follow the passive protocol described above and reconstruct  $\mu_\gamma$  to  $P_1$ .
- For (2), to be able to compute  $\langle \Delta \cdot \mu_{\alpha'_i} \rangle$  for some wire  $\alpha'_i$  in the next layer, it is sufficient to let all parties hold  $\langle \Delta \cdot \mu_{\gamma_i} \rangle$  for all  $i \in \{1, \dots, k\}$ . To this end, we try to follow the procedure of computing  $[\mu_\gamma]_{n-1}$ . Recall that

$$\begin{aligned} [\mu_\gamma]_{n-1} &= [\mathbf{v}_\alpha - \mathbf{a}]_{k-1} * [\mathbf{v}_\beta - \mathbf{b}]_{k-1} + [\mathbf{v}_\alpha - \mathbf{a}]_{k-1} * [\mathbf{b}]_{n-k} \\ &\quad + [\mathbf{v}_\beta - \mathbf{b}]_{k-1} * [\mathbf{a}]_{n-k} + [\mathbf{c}]_{n-1} - [\lambda_\gamma]_{n-1}. \end{aligned}$$

1. For  $[\mathbf{v}_\alpha - \mathbf{a}]_{k-1} * [\mathbf{b}]_{n-k}$  and  $[\mathbf{v}_\beta - \mathbf{b}]_{k-1} * [\mathbf{a}]_{n-k}$ , since all parties also hold  $[\Delta \cdot \mathbf{a}]_{n-k}$  and  $[\Delta \cdot \mathbf{b}]_{n-k}$ , they may locally compute  $[\mathbf{v}_\alpha - \mathbf{a}]_{k-1} * [\Delta \cdot \mathbf{b}]_{n-k}$  and  $[\mathbf{v}_\beta - \mathbf{b}]_{k-1} * [\Delta \cdot \mathbf{a}]_{n-k}$  and convert them locally to  $\langle \Delta \cdot (v_{\alpha_i} - a_i) \cdot b_i \rangle$  and  $\langle \Delta \cdot (v_{\beta_i} - b_i) \cdot a_i \rangle$ .
2. For  $[\mathbf{c}]_{n-1}$  and  $[\lambda_\gamma]_{n-1}$ , all parties already hold  $\langle \Delta \cdot c_i \rangle$  and  $\langle \Delta \cdot \lambda_{\gamma_i} \rangle$ .
3. The problematic part is to obtain  $\langle \Delta \cdot (v_{\alpha_i} - a_i) \cdot (v_{\beta_i} - b_i) \rangle$ . There we use the degree- $t$  Shamir sharing  $[\Delta|_i]_t$  as follows. We note that

$$[\Delta \cdot (v_{\alpha_i} - a_i) \cdot (v_{\beta_i} - b_i)]_i|_{n-1} = [\Delta|_i]_t * [\mathbf{v}_\alpha - \mathbf{a}]_{k-1} * [\mathbf{v}_\beta - \mathbf{b}]_{k-1}.$$

This follows from the multiplication of the underlying polynomials and the fact that  $n - 1 = t + 2(k - 1)$ . From  $[\Delta \cdot (v_{\alpha_i} - a_i) \cdot (v_{\beta_i} - b_i)]_i|_{n-1}$ , all parties can locally compute an additive sharing of  $\Delta \cdot (v_{\alpha_i} - a_i) \cdot (v_{\beta_i} - b_i)$ . Summing all terms up, all parties can *locally* obtain  $\langle \Delta \cdot \mu_{\gamma_i} \rangle$ .

- For (2), to be able to compute  $\langle \Delta \cdot (\lambda_{\alpha'_i} - a'_i) \rangle$  for some wire  $\alpha'_i$  in the next layer, it is sufficient to show how to obtain  $\langle \Delta \cdot \lambda_{\alpha'_i} \rangle$  since all parties can obtain  $\langle \Delta \cdot a'_i \rangle$  from  $[\Delta \cdot \mathbf{a}']_{n-k}$  prepared in the preprocessing data. Note that all parties already hold  $\langle \Delta \cdot \lambda_{\gamma_i} \rangle$  for the current layer. By following the circuit topology, they can locally compute  $\langle \Delta \cdot \lambda_{\alpha'_i} \rangle$  for the next layer.

*Checking the Correctness of the Computation.* All parties together hold additive sharings  $\langle \Delta \cdot \mu_{\alpha_i} \rangle$  and  $\langle \Delta \cdot (\lambda_{\alpha_i} - a_i) \rangle$ , they compute  $\langle \Delta \cdot (v_{\alpha_i} - a_i) \rangle$ . On the other hand, all parties hold a degree- $(k-1)$  packed Shamir sharing  $[\mathbf{v}_\alpha - \mathbf{a}]_{k-1}$ .

It is sufficient to check the following two points:

- The sharing  $[\mathbf{v}_\alpha - \mathbf{a}]_{k-1}$  is a valid degree- $(k-1)$  packed Shamir sharing. I.e., the shares lie on a degree- $(k-1)$  polynomial. The check is done by opening a random linear combination of all degree- $(k-1)$  packed Shamir sharings distributed by  $P_1$ .
- The secrets of  $[\mathbf{v}_\alpha - \mathbf{a}]_{k-1}$  are consistent with the MACs  $\{\langle \Delta \cdot (v_{\alpha_i} - a_i) \rangle\}_{i=1}^k$ . This is done by using  $[\mathbf{v}_\alpha - \mathbf{a}]_{k-1}$  and  $\{[\Delta|_i]_t\}_{i=1}^k$  to compute another version of MACs:  $\{\langle \Delta \cdot (v_{\alpha_i} - a_i) \rangle\}_{i=1}^k$ , and then check whether these two versions have the same secrets inside.

Both of these two checks are natural extensions of the checks done in SPDZ [14]. We thus omit the details and refer the readers to Sect. 4.4 for more details.

### 2.3 Instantiating the Circuit-Dependent Preprocessing

The preprocessing required by the parties is summarized as follows.

- A circuit-independent part, which are the global key  $[\Delta|_1]_t, \dots, [\Delta|_k]_t$  and a fresh packed Beaver triple with authentications per group of multiplication gates  $([\mathbf{a}]_{n-k}, [\Delta \cdot \mathbf{a}]_{n-k}), ([\mathbf{b}]_{n-k}, [\Delta \cdot \mathbf{b}]_{n-k}), ([\mathbf{c}]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k)$ .
- A circuit-dependent part that consists of  $[\lambda_\alpha]_{n-k}, [\lambda_\beta]_{n-k}, ([\lambda_\gamma]_{n-1}, \{\langle \Delta \cdot \lambda_{\gamma_i} \rangle\}_{i=1}^k)$ . Also  $P_1$  needs to obtain  $\lambda_\alpha - \mathbf{a}$  and  $\lambda_\beta - \mathbf{b}$ .

For the circuit-independent part, we will focus more on the preparation of the packed Beaver triples with authentications in the next section since the size of  $[\Delta|_1]_t, \dots, [\Delta|_k]_t$  is independent of the circuit size. As for the circuit-dependent part, we essentially follow the same idea in TURBOPACK [15] including the preprocessing data we need from a circuit-independent preprocessing, with the only exception that the preprocessing data should be authenticated. We refer the readers to [15] and Sect. 5 for more details.

*On the Necessity of a Circuit-Dependent Preprocessing.* At a first glance, it may appear that if the circuit only contain multiplication gates, then there is no need to have a circuit-dependent preprocessing phase since all  $\lambda$  values are uniform. We stress that this is not the case. This is because each wire  $\alpha$  is served as an output wire in a previous layer and then served as an input layer in a next layer. We need all parties to hold two packed Shamir sharings that contain  $\lambda_\alpha$ , one for a previous layer where  $\alpha$  is an output wire, and the other one for a next layer where  $\alpha$  is an input wire. In particular, the positions of  $\lambda_\alpha$  depend on the circuit topology since we need the two input packed Shamir sharings of a group of multiplication gates to have their secrets correctly aligned.

## 2.4 Instantiating the Circuit-Independent Preprocessing

Next, we focus on the preparation of authenticated packed Beaver triples:

$$([\mathbf{a}]_{n-k}, [\Delta \cdot \mathbf{a}]_{n-k}), ([\mathbf{b}]_{n-k}, [\Delta \cdot \mathbf{b}]_{n-k}), ([\mathbf{c}]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k),$$

where  $\mathbf{c} = \mathbf{a} * \mathbf{b}$ .

To this end, we make use of two functionalities  $\mathcal{F}_{\text{nVOLE}}$  and  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  from [25]. In [25], these two functionalities are used to efficiently prepare Beaver triples using additive sharings. At a high level,

1. All parties first use  $\mathcal{F}_{\text{nVOLE}}$  to prepare authenticated random additive sharings. In particular,
  - All parties receive an additive sharing  $\langle \Delta \rangle = (\Delta^1, \dots, \Delta^n)$  from  $\mathcal{F}_{\text{nVOLE}}$ , where  $\Delta$  is served as the MAC key. (Here  $\Delta^i$  is the  $i$ -th share of  $\langle \Delta \rangle$ .)
  - Each party  $P_i$  receives a vector  $\mathbf{u}^i$ , which is served as the additive shares held by  $P_i$ . We denote the additive sharings by  $\langle u_1 \rangle, \dots, \langle u_m \rangle$ .
  - For every ordered pair  $(P_i, P_j)$ , they together hold an additive sharing of  $\mathbf{u}^i \cdot \Delta^j$ . From these, all parties locally transform them to additive sharings  $\langle \Delta \cdot u_1 \rangle, \dots, \langle \Delta \cdot u_m \rangle$ .
2. After using  $\mathcal{F}_{\text{nVOLE}}$  to prepare two vectors of additive sharings, say  $(\langle a_1 \rangle, \langle b_1 \rangle), \dots, (\langle a_m \rangle, \langle b_m \rangle)$  together with their MACs, every ordered pair of parties  $(P_i, P_j)$  invokes  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  to compute additive sharings of  $a_\ell^i \cdot b_\ell^j$  for all  $\ell \in \{1, \dots, m\}$ . (Here  $a_\ell^i$  is the  $i$ -th share of  $\langle a_\ell \rangle$  and  $b_\ell^j$  is the  $j$ -th share of  $\langle b_\ell \rangle$ .) These allow all parties to obtain additive sharings of  $\mathbf{c} = (a_1 \cdot b_1, \dots, a_m \cdot b_m)$ . Note that the MACs of  $\langle c_1 \rangle, \dots, \langle c_m \rangle$  are *not* computed in this step.
3. Finally, all parties authenticate  $\langle c_1 \rangle, \dots, \langle c_m \rangle$  by using random additive sharings  $(\langle r_1 \rangle, \dots, \langle r_m \rangle)$  with authentications which can be prepared using Step 1.

As we discussed above, one direct solution would be using the above approach in a black box way and then transforming additive sharings to packed Shamir sharings. However, the direct consequence is that we need to not only pay the same cost as that in [25], but pay the additional cost for the sharing transformation as well. In the following we discuss how to take the advantage of the constant fraction of honest parties when preparing packed Beaver triples.

*Obtaining Authenticated Shares*  $([\mathbf{a}]_{n-k}, [\Delta \cdot \mathbf{a}]_{n-k})$ . We first discuss how the parties can obtain  $[\mathbf{a}]_{n-k}$  and  $[\Delta \cdot \mathbf{a}]_{n-k}$  (and also  $[\mathbf{b}]_{n-k}$  and  $[\Delta \cdot \mathbf{b}]_{n-k}$ ).

Our main observation is that the shares of a random degree- $(n-1)$  packed Shamir sharing are uniformly distributed. This is because a random degree- $(n-1)$  packed Shamir sharing corresponds to a random degree- $(n-1)$  polynomial, which satisfies that any  $n$  evaluations are uniformly distributed. On the other hand, the shares of a random additive sharing are also uniformly distributed. Thus, we may naturally view the random additive sharings prepared in  $\mathcal{F}_{\text{nVOLE}}$  as degree- $(n-1)$  packed Shamir sharings. Concretely, for each random additive sharing  $(u^1, \dots, u^n)$ , let  $\mathbf{u}$  denote the secrets of the degree- $(n-1)$  packed Shamir sharing when the shares are  $(u^1, \dots, u^n)$ . Then we may view that all parties hold the

packed Shamir sharing  $[\mathbf{u}]_{n-1}$ . To obtain a degree- $(n-k)$  packed Shamir sharing of  $\mathbf{u}$ , we simply perform a sharing transformation via the standard “mask-open-unmask” approach following from the known techniques [13].

Now the problem is to prepare the MACs for  $\mathbf{u}$ . We observe that in  $\mathcal{F}_{\text{nVOLE}}$ , for every ordered pair of parties  $(P_i, P_j)$ ,  $P_i, P_j$  together hold an additive sharing of  $u^i \cdot \Delta^j$ . Since each secret  $u_\ell$  in  $\mathbf{u}$  is a linear combination of  $(u^1, \dots, u^n)$ , all parties can locally compute an additive sharing of  $u_\ell \cdot \Delta^j$  for each  $j \in \{1, \dots, n\}$  and then compute an additive sharing of  $\Delta \cdot u_\ell$ . To obtain the MACs  $[\Delta \cdot \mathbf{u}]_{n-k}$ , we will perform a sharing transformation again via the standard “mask-open-unmask” approach following from the known techniques [13, 22].

In this way, to obtain a pair of authenticated sharings  $([\mathbf{a}]_{n-k}, [\Delta \cdot \mathbf{a}]_{n-k})$ , we only need to perform once the transformation from additive sharings to packed Shamir sharings. In addition, we essentially obtain such a pair of authenticated sharing from the same data that is only for one authenticated additive sharing in [25]. As a result, the amount of preprocessing data we need from  $\mathcal{F}_{\text{nVOLE}}$  is reduced by a factor of  $k = \epsilon n/2$ .

*Authenticated Product*  $([\mathbf{c}]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k)$ . Once the parties have obtained  $([\mathbf{a}]_{n-k}, [\Delta \cdot \mathbf{a}]_{n-k})$  and  $([\mathbf{b}]_{n-k}, [\Delta \cdot \mathbf{b}]_{n-k})$ , they need to obtain  $([\mathbf{c}]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k)$ , where  $\mathbf{c} = \mathbf{a} * \mathbf{b}$ .

To this end, we need to reuse the degree- $(n-1)$  packed Shamir sharings  $[\mathbf{a}]_{n-1}$  and  $[\mathbf{b}]_{n-1}$  output by  $\mathcal{F}_{\text{nVOLE}}$ . As that in [25], every ordered pair of parties  $(P_i, P_j)$  invokes  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  to compute additive sharings of  $a^i \cdot b^j$ , where  $a^i$  is the  $i$ -th share of  $[\mathbf{a}]_{n-1}$  and  $b^j$  is the  $j$ -th share of  $[\mathbf{b}]_{n-1}$ . From additive sharings of  $\{a^i \cdot b^j\}_{i,j}$ , all parties can locally compute an additive sharing of each  $c_\ell = a_\ell \cdot b_\ell$  for all  $\ell \in \{1, \dots, k\}$ . Finally, we obtain  $[\mathbf{c}]_{n-1}$  with authentications by using random sharings  $([\mathbf{r}]_{n-1}, \{\langle \Delta \cdot r_\ell \rangle\}_{\ell=1}^k)$  and follow the standard “mask-open-unmask” approach. Note that  $([\mathbf{r}]_{n-1}, \{\langle \Delta \cdot r_\ell \rangle\}_{\ell=1}^k)$  can be directly obtained from  $\mathcal{F}_{\text{nVOLE}}$  by properly interpreting the output of  $\mathcal{F}_{\text{nVOLE}}$  as we discussed above.

Thus, to prepare the authenticated product  $([\mathbf{c}]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k)$ , we only need to perform once the transformation from additive sharings to packed Shamir sharings. Again the amount of preprocessing data we need from  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  is also reduced by a factor of  $k = \epsilon n/2$ .

*Remarks About Our Techniques.* Note that we essentially follow the same steps as those in [25] but interpreting the output differently, and then perform sharing transformations to obtain sharings in the desired form. We would like to point out that *following the same steps as those in [25]* is crucial since in [25],  $\mathcal{F}_{\text{nVOLE}}$  only outputs random seeds to parties and the parties need to compute their shares by locally expanding the seeds using a proper PRG. And the same seeds are fed in  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  to compute the product sharings. Only in this way together with proper realizations of  $\mathcal{F}_{\text{nVOLE}}$  and  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$ , [25] can achieve sub-linear communication complexity in preparing Beaver triples (without authenticating the product sharing  $\langle c \rangle$ ). Thus, to be able to properly use the functionalities in [25], we should follow a similar pattern to that in [25].



*Verification of Packed Beaver Triples.* We note that the packed Beaver triples we obtained may be incorrect. This is because the invocations of  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  are between every pair of parties and the functionality  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  does not force the same party to use the same input across different invocations. Also when the product sharings are authenticated, corrupted parties may introduce additive errors. The same issues also appear in [25].

To obtain correct packed Beaver triples with authentications, our idea is to extend the technique of sacrificing [12] and use one possibly incorrect packed Beaver triple to check another possibly incorrect packed Beaver triple. To improve the concrete efficiency, we show that it is sufficient to have the sacrificed packed Beaver triple prepared in the form:

$$([\tilde{\mathbf{a}}]_{n-1}, \{\langle \Delta \cdot \tilde{a}_i \rangle\}_{i=1}^k), ([\tilde{\mathbf{b}}]_{n-1}, \{\langle \Delta \cdot \tilde{b}_i \rangle\}_{i=1}^k), ([\tilde{\mathbf{c}}]_{n-1}, \{\langle \Delta \cdot \tilde{c}_i \rangle\}_{i=1}^k).$$

I.e., we do not need to do any sharing transformation for the first two pairs of sharings and only need to authenticate the product sharing. We defer the details to the full version of this paper due to space constraints.

### 3 Preliminaries

**The Model.** We consider the task of secure multiparty computation in the client-server model, where a set of clients  $\mathcal{C} = \{C_1, \dots, C_m\}$  provide inputs to a set of computing parties  $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ , who carry out the computation and return output to the clients. Clients are connected to parties, and parties are connected to each other using a secure (private and authentic) synchronous channel. The communication complexity is measured by the total number of bits via private channels.

We focus on functions which can be represented as an arithmetic circuit  $C$  over a finite field  $\mathbb{F}$  with input, addition, multiplication, and output gates.<sup>4</sup> The circuit  $C$  takes inputs  $(\mathbf{x}_1, \dots, \mathbf{x}_m)$  and returns  $(\mathbf{y}_1, \dots, \mathbf{y}_m)$ , where  $\mathbf{x}_i \in \mathbb{F}^{l_i}$  and  $\mathbf{y}_i \in \mathbb{F}^{o_i}$ , for  $i \in \{1, \dots, m\}$ . We use the convention of labeling wires by means of greek letters (e.g.  $\alpha, \beta, \gamma$ ), and we use  $v_\alpha$  to denote the value stored in a wire labeled by  $\alpha$  for a given execution. We use  $\kappa$  to denote the security parameter, and we assume that  $|\mathbb{F}| \geq 2^\kappa$ . We assume that the number of parties  $n$  and the circuit size  $|C|$  are bounded by polynomials of the security parameter  $\kappa$ .

We study the dishonest majority setting where the adversary corrupts a majority of the parties, but we focus on the case where the number of corruptions may not be equal to  $n - 1$ . Instead, the adversary corrupts  $t < n(1 - \epsilon)$  parties for some constant  $0 < \epsilon < 1/2$ . For security we use Canetti's UC framework [8], where security is argued by the indistinguishability of an ideal world, modeled by

<sup>4</sup> In this work, we only focus on deterministic functions. A randomized function can be transformed into a deterministic function by taking as input an additional random tape from each party. The XOR of the input random tapes of all parties is used as the randomness of the randomized function.

a *functionality* (denoted in this work by the letter  $\mathcal{F}$  and some subscript), and the real world, instantiated by a *protocol* (denoted using the letter  $\Pi$  and some subscript). Protocols can also use *procedures*, denoted using the lowercase letter  $\pi$  and some subscript, which are like protocols except they are not intended to instantiate a given functionality, and instead they are used as “macros” inside other protocols that instantiate some functionality. The details on the security definition will be included in the full version of this paper.

We denote by  $\mathcal{F}_{\text{MPC}}$  the functionality that receives inputs from the clients, evaluates the function  $f$ , and returns output to the clients. This is given in detail in the full version of this paper. Security with unanimous abort, where all honest parties may jointly abort in the computation, is the best that can be achieved in the dishonest majority setting. Here we achieve security with selective abort, where the adversary can choose which honest parties abort, which can be compiled to unanimous abort using a broadcast channel [19]. To accommodate for aborts, every functionality in this work implicitly allows the adversary to send an abort signal to a specific honest party. We do not write this explicitly.

**Packed Shamir Secret Sharing.** In our work, we make use of packed Shamir secret sharing, introduced by Franklin and Yung [16]. This is a generalization of the standard Shamir secret sharing scheme [26]. Let  $n$  be the number of parties and  $k$  be the number of secrets to pack in one sharing. A *degree- $d$*  ( $d \geq k - 1$ ) packed Shamir sharing of  $\mathbf{x} = (x_1, \dots, x_k) \in \mathbb{F}^k$  is a vector  $(w_1, \dots, w_n)$  for which there exists a polynomial  $f(\cdot) \in \mathbb{F}[X]$  of degree at most  $d$  such that  $f(-i + 1) = x_i$  for all  $i \in \{1, 2, \dots, k\}$ , and  $f(i) = w_i$  for all  $i \in \{1, 2, \dots, n\}$ . The  $i$ -th share  $w_i$  is held by party  $P_i$ . Reconstructing a degree- $d$  packed Shamir sharing requires  $d + 1$  shares and can be done by Lagrange interpolation. For a random degree- $d$  packed Shamir sharing of  $\mathbf{x}$ , any  $d - k + 1$  shares are independent of the secret  $\mathbf{x}$ . If  $d - (k - 1) \geq t$ , then knowing  $t$  of the shares does not leak anything about the  $k$  secrets. In particular, a sharing of degree  $t + (k - 1)$  keeps hidden the underlying  $k$  secret.

In our work, we use  $[\mathbf{x}]_d$  to denote a degree- $d$  packed Shamir sharing of  $\mathbf{x} \in \mathbb{F}^k$ . In the following, operations (addition and multiplication) between two packed Shamir sharings are coordinate-wise, and  $*$  denotes element-wise product. We recall two properties of the packed Shamir sharing scheme:

- **Linear Homomorphism:** For all  $d \geq k - 1$  and  $\mathbf{x}, \mathbf{y} \in \mathbb{F}^k$ ,  $[\mathbf{x} + \mathbf{y}]_d = [\mathbf{x}]_d + [\mathbf{y}]_d$ .
- **Multiplicativity:** Let  $*$  denote the coordinate-wise multiplication operation. For all  $d_1, d_2 \geq k - 1$  subject to  $d_1 + d_2 < n$ , and for all  $\mathbf{x}, \mathbf{y} \in \mathbb{F}^k$ ,  $[\mathbf{x} * \mathbf{y}]_{d_1 + d_2} = [\mathbf{x}]_{d_1} * [\mathbf{y}]_{d_2}$ .

Note that the second property implies that, for all  $\mathbf{x}, \mathbf{c} \in \mathbb{F}^k$ , all parties can locally compute  $[\mathbf{c} * \mathbf{x}]_{d+k-1}$  from  $[\mathbf{x}]_d$  and the public vector  $\mathbf{c}$ . To see this, all parties can locally transform  $\mathbf{c}$  to a degree- $(k - 1)$  packed Shamir sharing  $[\mathbf{c}]_{k-1}$ . Then, they can use the property of the packed Shamir sharing scheme to compute  $[\mathbf{c} * \mathbf{x}]_{d+k-1} = [\mathbf{c}]_{k-1} * [\mathbf{x}]_d$ . We simply write  $[\mathbf{c} * \mathbf{x}]_{d+k-1} = \mathbf{c} * [\mathbf{x}]_d$  to denote this procedure.

When the packing parameter  $k = 1$ , a packed Shamir sharing degrades to a Shamir sharing. Generically, a Shamir sharing uses the default evaluation point 0 to store the secret. In our work, we are interested in using different evaluation points in different Shamir secret sharings. Concretely, for all  $i \in \{1, \dots, k\}$ , we use  $[x|_i]_d$  to represent a degree- $d$  Shamir sharing of  $x$  such that the secret is stored at the evaluation point  $-i + 1$ . If we use  $f$  to denote the degree- $d$  polynomial corresponding to  $[x|_i]_d$ , then  $f(-i + 1) = x$ .

In this work, we choose the packing parameter to be  $k = (n - t + 1)/2$  (assume for simplicity that this division is exact), or equivalently  $n = t + 2k - 1 = t + 2(k - 1) + 1$ . This implies not only that a sharing of degree  $t + (k - 1)$  (which keeps the privacy of  $k$  secrets) is well defined as there are more parties than the degree plus one, but also if a sharing of such degree is multiplied by a degree- $(k - 1)$  sharing, the resulting degree- $(t + 2(k - 1))$  sharing is also well defined. Also, we observe that with these parameters, a sharing of degree at most  $2(k - 1)$  is fully determined by the honest parties' shares since  $n - t = 2(k - 1) + 1$ , which in particular means that such sharings can be reconstructed to obtain the correct underlying secrets (*i.e.* the secrets determined by the honest parties' shares). Finally, recall that  $t < n(1 - \epsilon)$ . We assume that  $t + 1 = (1 - \epsilon)n$  for simplicity, and in this case it can be checked that  $k = \frac{\epsilon}{2} \cdot n + 1 = \Theta(n)$ .

**Some Functionalities.** For our protocols we assume the existence of two widely used functionalities. One is  $\mathcal{F}_{\text{Coin}}$ , which upon being called provides the parties with a uniformly random value  $r \in \mathbb{F}$ . This can be easily implemented by having the parties open some random shared value  $\langle r \rangle$ , and if more coins are needed these can be expanded with the help of a PRG. The second functionality is  $\mathcal{F}_{\text{Commit}}$ , which enables the parties to commit to some values of their choice without revealing them to the other parties. At a later point, the parties can open their committed values with the guarantee that these opened terms are exactly the same that were committed to initially. This can be instantiated with the help of a hash function, modeled as a random oracle (*cf.* [12]).

## 4 Online Protocol

We begin by describing the online phase of SUPERPACK.

### 4.1 Circuit-Dependent Preprocessing Functionality

In order to securely compute the given function, our online phase must make use of certain *circuit-dependent* preprocessing, which is modeled in Functionality  $\mathcal{F}_{\text{PrepMal}}$  below.

### Functionality 1: $\mathcal{F}_{\text{PrepMal}}$

1. **Assign Random Values to Wires in  $C$ :**  $\mathcal{F}_{\text{PrepMal}}$  receives the circuit  $C$  from all parties. Then  $\mathcal{F}_{\text{PrepMal}}$  assigns random values to wires in  $C$  as follows.
  - (a) For each output wire  $\alpha$  of an input gate or a multiplication gate,  $\mathcal{F}_{\text{PrepMal}}$  samples a uniform value  $\lambda_\alpha$  and associates it with the wire  $\alpha$ .
  - (b) Starting from the first layer of  $C$  to the last layer, for each addition gate with input wires  $\alpha, \beta$  and output wire  $\gamma$ ,  $\mathcal{F}_{\text{PrepMal}}$  sets  $\lambda_\gamma = \lambda_\alpha + \lambda_\beta$ .
2. **Settling Authentication Keys:**  $\mathcal{F}_{\text{PrepMal}}$  samples a random value  $\Delta$ . Then  $\mathcal{F}_{\text{PrepMal}}$  samples  $k$  random degree- $t$  Shamir sharings  $([\Delta]_1|_t, \dots, [\Delta]_k|_t)$  and distributes the shares to all parties.
3. **Preparing Packed Beaver Triples with Authentications:** For each group of  $k$  multiplication gates,  $\mathcal{F}_{\text{PrepMal}}$  samples a random packed Beaver triple with authentications as follows:
  - (a)  $\mathcal{F}_{\text{PrepMal}}$  samples two random vectors  $\mathbf{a}, \mathbf{b} \in \mathbb{F}_p^k$  and computes  $\Delta \cdot \mathbf{a}, \Delta \cdot \mathbf{b}$ . Then  $\mathcal{F}_{\text{PrepMal}}$  samples two pairs of random degree- $(n-k)$  packed Shamir sharings  $([\mathbf{a}]_{n-k} = ([\mathbf{a}]_{n-k}, [\Delta \cdot \mathbf{a}]_{n-k}), [\mathbf{b}]_{n-k} = ([\mathbf{b}]_{n-k}, [\Delta \cdot \mathbf{b}]_{n-k}))$ .
  - (b)  $\mathcal{F}_{\text{PrepMal}}$  computes  $\mathbf{c} = \mathbf{a} * \mathbf{b}$  and  $\Delta \cdot \mathbf{c}$ . Then  $\mathcal{F}_{\text{PrepMal}}$  samples a random degree- $(n-1)$  packed Shamir sharing  $[c]_{n-1}$ . For all  $i \in \{1, \dots, k\}$ ,  $\mathcal{F}_{\text{PrepMal}}$  samples a random additive sharing  $\langle \Delta \cdot c_i \rangle$ .

$\mathcal{F}_{\text{PrepMal}}$  distributes the shares of  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, ([c]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$  to all parties.
4. **Distributing  $\lambda_\alpha - \mathbf{a}$  and  $\lambda_\beta - \mathbf{b}$  to  $P_1$ :** For each group of multiplication gates, let  $\alpha, \beta$  denote the batch of first input wires and that of the second input wires respectively. Let  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, ([c]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$  be the packed Beaver triple with authentications associated with these gates.  $\mathcal{F}_{\text{PrepMal}}$  receives two vectors of additive errors  $\delta_\alpha, \delta_\beta$  from the adversary, computes  $\lambda_\alpha - \mathbf{a} + \delta_\alpha$  and  $\lambda_\beta - \mathbf{b} + \delta_\beta$ , and sends them to  $P_1$ . Here  $\lambda_\alpha$  and  $\lambda_\beta$  are the random values associated with the wires  $\alpha$  and  $\beta$ .  $\mathcal{F}_{\text{PrepMal}}$  also samples random additive sharings  $\{\langle \Delta \cdot (\lambda_{\alpha_i} - a_i) \rangle, \langle \Delta \cdot (\lambda_{\beta_i} - b_i) \rangle\}_{i=1}^k$  and distributes the shares to all parties.
5. **Preparing Authenticated Packed Sharings for Multiplication Gates:** For each group of multiplication gates with output wires  $\gamma$ ,  $\mathcal{F}_{\text{PrepMal}}$  samples
  - A random degree- $(n-1)$  packed Shamir sharing  $[\lambda_\gamma]_{n-1}$ ,
  - $k$  additive sharings  $\{\langle \Delta \cdot \lambda_{\gamma_i} \rangle\}_{i=1}^k$ ,
 and distributes the shares to honest parties.
6. **Preparing Random Sharings for Input and Output Gates:** For each group of  $k$  input gates or output gates,  $\mathcal{F}_{\text{PrepMal}}$  prepares the following random sharings.
  - (a) Let  $\alpha$  be the output wires of these  $k$  input gates or the input wires of these  $k$  output gates.  $\mathcal{F}_{\text{PrepMal}}$  samples
    - A random degree- $(n-1)$  packed Shamir sharing  $[\lambda_\alpha]_{n-1}$ ,
    - $k$  additive sharings  $\{\langle \Delta \cdot \lambda_{\alpha_i} \rangle\}_{i=1}^k$ ,
 and distributes the shares to honest parties.
  - (b)  $\mathcal{F}_{\text{PrepMal}}$  also prepares a random packed Beaver triple with authentications  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, ([c]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$  in the same way as Step 3. Later, we will view  $\mathbf{b}$  as an authentication key and  $\mathbf{c}$  as the MAC of  $\mathbf{a}$ . This allows the input holder to verify the correctness of  $\mathbf{a}$ .

**Corrupted Parties:** When  $\mathcal{F}_{\text{PrepMal}}$  prepares random sharings, corrupted parties can choose their shares.  $\mathcal{F}_{\text{PrepMal}}$  then samples the random sharings based on the secret it generated and the shares chosen by the corrupted parties.

## 4.2 Input Gates

In this section, we give the description of the procedure  $\pi_{\text{Input}}$ . This procedure enables  $P_1$  to learn  $\mu_\alpha = v_\alpha - \lambda_\alpha$  for every input wire  $\alpha$ , where  $v_\alpha$  is the input provided by the client owning the input gate. In addition, the parties output shares of the MAC of this value, namely  $\{\langle \Delta \cdot \mu_{\alpha_i} \rangle\}_{i=1}^k$ . Recall that in  $\mathcal{F}_{\text{PrepMal}}$ , we prepared a packed Beaver triple with authentications  $(\llbracket \mathbf{a} \rrbracket_{n-k}, \llbracket \mathbf{b} \rrbracket_{n-k}, (\llbracket \mathbf{c} \rrbracket_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$  for each group of input gates. Here  $\mathbf{b}$  serves as the MAC key and  $\mathbf{c}$  serves as the MAC of  $\mathbf{a}$  so that the client can verify that he receives the correct  $\mathbf{a}$  in  $\pi_{\text{Input}}$ . The description of  $\pi_{\text{Input}}$  appears below.

### Procedure 1: $\pi_{\text{Input}}$

1. For each group of input gates that belongs to **Client**, let  $\alpha$  denote the batch of output wires of these input gates. All parties receive from  $\mathcal{F}_{\text{PrepMal}}$ 
  - A random degree- $(n-1)$  packed Shamir sharing  $[\lambda_\alpha]_{n-1}$  with MACs  $\{\langle \Delta \cdot \lambda_{\alpha_i} \rangle\}_{i=1}^k$ .
  - A packed Beaver triple with authentications  $(\llbracket \mathbf{a} \rrbracket_{n-k}, \llbracket \mathbf{b} \rrbracket_{n-k}, (\llbracket \mathbf{c} \rrbracket_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$ .

Let  $\mathbf{v}_\alpha$  denote the inputs held by **Client**.
2. All parties send to **Client** their shares of  $[\lambda_\alpha]_{n-1}, [\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, [\mathbf{c}]_{n-1}$ .
3. **Client** reconstructs the secrets  $\lambda_\alpha, \mathbf{a}, \mathbf{b}, \mathbf{c}$  and checks whether  $\mathbf{c} = \mathbf{a} * \mathbf{b}$ . If not, **Client** aborts. Otherwise, **Client** computes  $\mu_\alpha = \mathbf{v}_\alpha - \lambda_\alpha$  and  $[\mathbf{v}_\alpha - \mathbf{a}]_{2k-2}$ .
4. **Client** sends  $\mu_\alpha$  to  $P_1$  and distributes the shares of  $[\mathbf{v}_\alpha - \mathbf{a}]_{2k-2}$  to all parties.
5. For all  $i \in \{1, \dots, k\}$ , all parties locally compute  $\langle \Delta \cdot \mu_{\alpha_i} \rangle$  as follows:
  - (a) Recall that all parties hold  $[\Delta|_i]_t$  generated in  $\mathcal{F}_{\text{PrepMal}}$ . All parties locally compute  $[\Delta \cdot (v_{\alpha_i} - a_i)]_i|_{n-1} = [\Delta|_i]_t * [\mathbf{v}_\alpha - \mathbf{a}]_{2k-2}$ . Then all parties locally transform it to an additive sharing  $\langle \Delta \cdot (v_{\alpha_i} - a_i) \rangle$ .
  - (b) Recall that all parties hold  $[\Delta \cdot \mathbf{a}]_{n-k}$ . All parties locally transform it to an additive sharing  $\langle \Delta \cdot a_i \rangle$ .
  - (c) Recall that all parties hold  $\langle \Delta \cdot \lambda_{\alpha_i} \rangle$ . All parties locally compute  $\langle \Delta \cdot \mu_{\alpha_i} \rangle = \langle \Delta \cdot (v_{\alpha_i} - a_i) \rangle + \langle \Delta \cdot a_i \rangle - \langle \Delta \cdot \lambda_{\alpha_i} \rangle$ .

## 4.3 Computing Addition and Multiplication Gates

After receiving the inputs from all clients, all parties start to evaluate the circuit gate by gate. We will maintain the invariant that for each output wire  $\alpha$  of an input gate or a multiplication gate,  $P_1$  learns  $\mu_\alpha$  in clear. In the procedure,  $P_1$  distributes shares of certain values, which may be incorrect. To prevent cheating, the parties get additive shares of the MAC of these values, which are used in a verification step in the output phase to check for correctness.

**Procedure 2:**  $\pi_{\text{Mult}}$ 

The procedure is executed for a group of  $k$  multiplication gates with input wires  $\alpha$  and  $\beta$ , and output wires  $\gamma$ .

1. All parties hold
  - A packed Beaver triple with authentications  $(\llbracket \mathbf{a} \rrbracket_{n-k}, \llbracket \mathbf{b} \rrbracket_{n-k}, (\llbracket \mathbf{c} \rrbracket_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$ .
  - A random degree- $(n-1)$  packed Shamir sharing  $\llbracket \lambda_\gamma \rrbracket_{n-1}$  with MACs  $\{\langle \Delta \cdot \lambda_{\gamma_i} \rangle\}_{i=1}^k$ .
  - Additive sharings  $\{\langle \Delta \cdot (\lambda_{\alpha_i} - a_i) \rangle, \langle \Delta \cdot (\lambda_{\beta_i} - b_i) \rangle\}_{i=1}^k$ .
 And  $P_1$  learns
  - $\mu_\alpha = \mathbf{v}_\alpha - \lambda_\alpha$ ,  $\mu_\beta = \mathbf{v}_\beta - \lambda_\beta$  from the previous layers;
  - $\lambda_\alpha - \mathbf{a}$ ,  $\lambda_\beta - \mathbf{b}$  received from  $\mathcal{F}_{\text{PrepMal}}$ .
2.  $P_1$  locally computes  $\mathbf{v}_\alpha - \mathbf{a} = \mu_\alpha + \lambda_\alpha - \mathbf{a}$ . Similarly,  $P_1$  locally computes  $\mathbf{v}_\beta - \mathbf{b}$ . Then  $P_1$  distributes shares of  $[\mathbf{v}_\alpha - \mathbf{a}]_{k-1}$  and  $[\mathbf{v}_\beta - \mathbf{b}]_{k-1}$  to all parties.
3. For all  $i \in \{1, \dots, k\}$ , all parties locally compute  $\langle \theta_{\alpha_i} \rangle$  and  $\langle \theta_{\beta_i} \rangle$  as follows.
  - (a) Recall that all parties have computed additive sharings of the MACs of the  $\mu$  values for output wires of multiplication gates and input gates in previous layers. By using these additive sharings, all parties locally compute  $\langle \Delta \cdot \mu_{\alpha_i} \rangle$ ,  $\langle \Delta \cdot \mu_{\beta_i} \rangle$ .
  - (b) Recall that all parties hold  $\langle \Delta \cdot (\lambda_{\alpha_i} - a_i) \rangle$ ,  $\langle \Delta \cdot (\lambda_{\beta_i} - b_i) \rangle$ . They locally compute  $\langle \Delta \cdot (v_{\alpha_i} - a_i) \rangle = \langle \Delta \cdot \mu_{\alpha_i} \rangle + \langle \Delta \cdot (\lambda_{\alpha_i} - a_i) \rangle$  and  $\langle \Delta \cdot (v_{\beta_i} - b_i) \rangle = \langle \Delta \cdot \mu_{\beta_i} \rangle + \langle \Delta \cdot (\lambda_{\beta_i} - b_i) \rangle$ .
  - (c) Also recall that all parties hold  $[\Delta|_i]_t$ . All parties locally compute  $[\Delta|_i]_t * [\mathbf{v}_\alpha - \mathbf{a}]_{k-1}$  and transform it to an additive sharing  $\langle \Delta \cdot (v_{\alpha_i} - a_i) \rangle$ . Similarly, all parties locally compute  $[\Delta|_i]_t * [\mathbf{v}_\beta - \mathbf{b}]_{k-1}$  and transform it to an additive sharing  $\langle \Delta \cdot (v_{\beta_i} - b_i) \rangle$ .
  - (d) All parties locally compute  $\langle \theta_{\alpha_i} \rangle = \langle \Delta \cdot (v_{\alpha_i} - a_i) \rangle - \langle \Delta \cdot (v_{\alpha_i} - a_i) \rangle$  and  $\langle \theta_{\beta_i} \rangle = \langle \Delta \cdot (v_{\beta_i} - b_i) \rangle - \langle \Delta \cdot (v_{\beta_i} - b_i) \rangle$ .
4. All parties locally compute  $[\mu_\gamma]_{n-1} = [\mathbf{v}_\alpha - \mathbf{a}]_{k-1} * [\mathbf{v}_\beta - \mathbf{b}]_{k-1} + [\mathbf{v}_\alpha - \mathbf{a}]_{k-1} * [\mathbf{b}]_{n-k} + [\mathbf{v}_\beta - \mathbf{b}]_{k-1} * [\mathbf{a}]_{n-k} + [\mathbf{c}]_{n-1} - [\lambda_\gamma]_{n-1}$ .
5. For all  $i \in \{1, \dots, k\}$ , all parties locally compute an additive sharing  $\langle \Delta \cdot \mu_{\gamma_i} \rangle$  as follows.
  - (a) Recall that all parties hold  $[\Delta|_i]_t$  from  $\mathcal{F}_{\text{PrepMal}}$ . All parties locally compute  $[\Delta|_i]_t * [\mathbf{v}_\alpha - \mathbf{a}]_{k-1} * [\mathbf{v}_\beta - \mathbf{b}]_{k-1}$  and transform it to an additive sharing  $\langle \Delta \cdot (v_{\alpha_i} - a_i) \cdot (v_{\beta_i} - b_i) \rangle$ .
  - (b) Recall that all parties hold  $[\Delta \cdot \mathbf{a}]_{n-k}$  and  $[\Delta \cdot \mathbf{b}]_{n-k}$ . All parties locally compute  $[\mathbf{v}_\alpha - \mathbf{a}]_{k-1} * [\Delta \cdot \mathbf{b}]_{n-k} + [\mathbf{v}_\beta - \mathbf{b}]_{k-1} * [\Delta \cdot \mathbf{a}]_{n-k}$  and transform it to an additive sharing  $\langle \Delta \cdot ((v_{\alpha_i} - a_i) \cdot b_i + (v_{\beta_i} - b_i) \cdot a_i) \rangle$ .
  - (c) Recall that all parties hold  $\langle \Delta \cdot c_i \rangle$  and  $\langle \Delta \cdot \lambda_{\gamma_i} \rangle$ . All parties locally compute  $\langle \Delta \cdot \mu_{\gamma_i} \rangle = \langle \Delta \cdot (v_{\alpha_i} - a_i) \cdot (v_{\beta_i} - b_i) \rangle + \langle \Delta \cdot ((v_{\alpha_i} - a_i) \cdot b_i + (v_{\beta_i} - b_i) \cdot a_i) \rangle + \langle \Delta \cdot c_i \rangle - \langle \Delta \cdot \lambda_{\gamma_i} \rangle$ .
6.  $P_1$  collects the whole sharing  $[\mu_\gamma]_{n-1}$  from all parties and reconstructs  $\mu_\gamma$ .

#### 4.4 Output Gates and Verification

At the end of the protocol, all parties together check the correctness of the computation. We first transform the output sharings to sharings that can be conveniently checked by clients. However, before reconstructing these outputs to the clients, the parties jointly verify the correctness of the computation by checking that (1) the sharings distributed by  $P_1$  in  $\pi_{\text{Mult}}$  have the correct degree  $\leq k-1$ , and (2) the underlying secrets are correct, for which the MACs computed in the online phase are used.

Due to space constraints, we describe the procedure  $\pi_{\text{Output}}$  in detail in the full version of this paper, including the computation of the output gates, the verification of the computation (degree and MAC check), and the reconstruction of the outputs.

#### 4.5 Full Online Protocol

Our final online protocol makes use of the procedures  $\pi_{\text{Input}}$  (Procedure 1, Sect. 4.2) to let the clients distribute their inputs,  $\pi_{\text{Mult}}$  (Procedure 2, Sect. 4.3) to process each group of  $k$  multiplication gates, and  $\pi_{\text{Output}}$  (Sect. 4.4) to verify the correctness of the computation and reconstruct output to the clients.  $\pi_{\text{Output}}$  and the online protocol  $\Pi_{\text{Online}}$  are presented in detail in the full version. We prove the following:

**Theorem 2.** *Let  $c$  denote the number of servers and  $n$  denote the number of parties (servers). For all  $0 < \epsilon \leq 1/2$ , protocol  $\Pi_{\text{Online}}$  instantiates Functionality  $\mathcal{F}_{\text{MPC}}$  in the  $\mathcal{F}_{\text{PrepMal}}$ -hybrid model, with statistical security against a fully malicious adversary who can control up to  $c$  clients and  $t = (1 - \epsilon)n$  parties (servers).*

**Communication Complexity of  $\Pi_{\text{Online}}$ .** Let  $I$  and  $O$  be the number of input wires and output wires, and assume that each client owns a number of input and output gates that is a multiple of  $k$ . We assume for simplicity that  $n$  divides each of these terms, and also that  $n$  divides the number of multiplication gates in each layer. Let us also denote by  $|C|$  the number of multiplication gates in the circuit  $C$ . The total communication complexity is given by  $\frac{4}{\epsilon} \cdot (I + O) + \frac{6}{\epsilon} \cdot |C|$ , ignoring small terms that are independent of  $I$ ,  $O$  and  $|C|$ .

### 5 Circuit-Dependent Preprocessing Phase

In this section, we discuss how to realize the ideal functionality for the circuit-dependent preprocessing phase,  $\mathcal{F}_{\text{PrepMal}}$ , presented as Functionality 1. Recall that  $k = (n - t + 1)/2$ . For simplicity, we only focus on the scenario where  $t \geq n/2$ .

We realize  $\mathcal{F}_{\text{PrepMal}}$  by using a circuit-independent functionality,  $\mathcal{F}_{\text{PrepIndMal}}$ , which is described below.

**Functionality 2:**  $\mathcal{F}_{\text{PreIndMal}}$

1. **Setting Authentication Keys:**  $\mathcal{F}_{\text{PreIndMal}}$  samples a random value  $\Delta$ . Then  $\mathcal{F}_{\text{PreIndMal}}$  samples  $k$  random degree- $t$  Shamir sharings  $([\Delta]_1|_t, \dots, [\Delta]_k|_t)$  and distributes the shares to all parties.
2. **Preparing Random Packed Sharings:** For each output wire  $\alpha$  of an input gate or a multiplication gate in the circuit  $C$ ,  $\mathcal{F}_{\text{PreIndMal}}$  samples a random value as  $\lambda_\alpha$  and computes  $\lambda_\alpha \cdot \mathbf{1}$ , where  $\mathbf{1} = (1, \dots, 1) \in \mathbb{F}^k$ . Then  $\mathcal{F}_{\text{PreIndMal}}$  samples
  - a random degree- $(n - k)$  packed Shamir sharing  $[\lambda_\alpha \cdot \mathbf{1}]_{n-k}$ ,
  - and a random additive sharing  $\langle \Delta \cdot \lambda_\alpha \rangle$ ,
 and distributes the shares to all parties.
3. **Preparing Packed Beaver Triples with Authentications:** For each group of  $k$  multiplication gates,  $\mathcal{F}_{\text{PreIndMal}}$  samples a random packed Beaver triple with authentications as follows:
  - (a)  $\mathcal{F}_{\text{PreIndMal}}$  samples two random vectors  $\mathbf{a}, \mathbf{b} \in \mathbb{F}_p^k$  and computes  $\Delta \cdot \mathbf{a}, \Delta \cdot \mathbf{b}$ . Then  $\mathcal{F}_{\text{PreIndMal}}$  samples two pairs of random degree- $(n - k)$  packed Shamir sharings  $[\mathbf{a}]_{n-k} = ([\mathbf{a}]_{n-k}, [\Delta \cdot \mathbf{a}]_{n-k}), [\mathbf{b}]_{n-k} = ([\mathbf{b}]_{n-k}, [\Delta \cdot \mathbf{b}]_{n-k})$ .
  - (b)  $\mathcal{F}_{\text{PreIndMal}}$  computes  $\mathbf{c} = \mathbf{a} * \mathbf{b}$  and  $\Delta \cdot \mathbf{c}$ . Then  $\mathcal{F}_{\text{PreIndMal}}$  samples a random degree- $(n - 1)$  packed Shamir sharing  $[c]_{n-1}$ . For all  $i \in \{1, \dots, k\}$ ,  $\mathcal{F}_{\text{PreIndMal}}$  samples a random additive sharing  $\langle \Delta \cdot c_i \rangle$ .  $\mathcal{F}_{\text{PreIndMal}}$  distributes the shares of  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, ([c]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$  to all parties.
4. **Preparing Random Masked Sharings for Multiplication Gates:** For each group of  $k$  multiplication gates,  $\mathcal{F}_{\text{PreIndMal}}$  sets  $\mathbf{o}^{(1)} = \mathbf{o}^{(2)} = \mathbf{o}^{(3)} = \mathbf{0} \in \mathbb{F}^k$ . Then  $\mathcal{F}_{\text{PreIndMal}}$  samples three random degree- $(n - 1)$  packed Shamir sharings  $[\mathbf{o}^{(1)}]_{n-1}, [\mathbf{o}^{(2)}]_{n-1}, [\mathbf{o}^{(3)}]_{n-1}$  and distributes the shares to all parties.
5. **Preparing Random Sharings for Input and Output Gates:** For each group of  $k$  input gates or output gates,  $\mathcal{F}_{\text{PreIndMal}}$  prepares the following random sharings.
  - (a)  $\mathcal{F}_{\text{PreIndMal}}$  prepares a random degree- $(n - 1)$  packed Shamir sharing of  $\mathbf{0} \in \mathbb{F}^k$ , denoted by  $[\mathbf{o}]_{n-1}$ , in the same way as Step 4.
  - (b)  $\mathcal{F}_{\text{PreIndMal}}$  also prepares a random packed Beaver triple with authentications  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, ([c]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$  in the same way as Step 3. Later, we will view  $\mathbf{b}$  as an authentication key and  $\mathbf{c}$  as the MAC of  $\mathbf{a}$ . This allows the input holder to verify the correctness of  $\mathbf{a}$ .

**Corrupted Parties:** When  $\mathcal{F}_{\text{PreIndMal}}$  prepares random sharings, corrupted parties can choose their shares.  $\mathcal{F}_{\text{PreIndMal}}$  then samples the random sharings based on the secret it generated and the shares chosen by the corrupted parties.

To instantiate the circuit-dependent preprocessing functionality  $\mathcal{F}_{\text{PreMal}}$  using the circuit-independent preprocessing  $\mathcal{F}_{\text{PreIndMal}}$ , we follow the idea in [15]. We describe the protocol  $\Pi_{\text{PreMal}}$  below.



**Protocol 3:**  $\Pi_{\text{PrepMal}}$ 

1. All parties invoke  $\mathcal{F}_{\text{PrepIndMal}}$ .
2. **Setting Authentication Keys:** All parties use  $\{[\Delta|_i]_t\}_{i=1}^k$  generated in  $\mathcal{F}_{\text{PrepIndMal}}$ .
3. **Preparing Packed Beaver Triples with Authentications:** All parties use the packed Beaver triples with authentications prepared in  $\mathcal{F}_{\text{PrepIndMal}}$ .
4. **Distributing  $\lambda_\alpha - \mathbf{a}$  and  $\lambda_\beta - \mathbf{b}$  to  $P_1$ :** In  $\mathcal{F}_{\text{PrepIndMal}}$ , for each output wire  $\alpha$  of an input gate or a multiplication gate, all parties obtain a random degree- $(n - k)$  packed Shamir sharing with authentication in the form of  $([\lambda_\alpha \cdot \mathbf{1}]_{n-k}, \langle \Delta \cdot \lambda_\alpha \rangle)$ . All parties locally compute  $([\lambda_\alpha \cdot \mathbf{1}]_{n-k}, \langle \Delta \cdot \lambda_\alpha \rangle)$  for every wire  $\alpha$  in the circuit.

For each group of multiplication gates, let  $\alpha, \beta$  denote the batch of first input wires and that of the second input wires respectively. Let  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, ([c]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$  be the packed Beaver triple with authentications associated with these gates. Let  $([\mathbf{o}^{(1)}]_{n-1}, [\mathbf{o}^{(2)}]_{n-1}, [\mathbf{o}^{(3)}]_{n-1})$  be the random degree- $(n - 1)$  packed Shamir sharings of  $\mathbf{0}$  prepared in  $\mathcal{F}_{\text{PrepIndMal}}$ . All parties run the following steps:

- (a) All parties locally compute  $[\lambda_\alpha - \mathbf{a}]_{n-1} = \left( \sum_{i=1}^k e_i * [\lambda_{\alpha_i} \cdot \mathbf{1}]_{n-k} \right) - [\mathbf{a}]_{n-k} + [\mathbf{o}^{(1)}]_{n-1}$  and send their shares to  $P_1$ .
- (b)  $P_1$  reconstructs  $\lambda_\alpha - \mathbf{a}$ .
- (c) For all  $i \in \{1, \dots, k\}$ , all parties locally transform  $[\Delta \cdot \mathbf{a}]_{n-k}$  to an additive sharing  $\langle \Delta \cdot a_i \rangle$ . Then all parties locally compute  $\langle \Delta \cdot (\lambda_{\alpha_i} - a_i) \rangle = \langle \Delta \cdot \lambda_{\alpha_i} \rangle - \langle \Delta \cdot a_i \rangle$ . All parties locally refresh the obtained additive sharing.<sup>a</sup> As a result, all parties hold a random additive sharing of  $\Delta \cdot (\lambda_{\alpha_i} - a_i)$ .
- (d) Repeat the above steps for  $\lambda_\beta - \mathbf{b}$  using  $[\mathbf{o}^{(2)}]_{n-1}$ .
5. **Preparing Authenticated Packed Sharings for Multiplication Gates:** For each group of multiplication gates with output wires  $\gamma$ , let  $([\mathbf{o}^{(1)}]_{n-1}, [\mathbf{o}^{(2)}]_{n-1}, [\mathbf{o}^{(3)}]_{n-1})$  be the random degree- $(n - 1)$  packed Shamir sharings of  $\mathbf{0}$  prepared in  $\mathcal{F}_{\text{PrepIndMal}}$ . Recall that all parties hold  $\{([\lambda_{\gamma_i} \cdot \mathbf{1}]_{n-k}, \langle \Delta \cdot \lambda_{\gamma_i} \rangle)\}_{i=1}^k$ . All parties locally compute  $[\lambda_\gamma]_{n-1} = \left( \sum_{i=1}^k e_i * [\lambda_{\gamma_i} \cdot \mathbf{1}]_{n-k} \right) + [\mathbf{o}^{(3)}]_{n-1}$ .
6. **Preparing Random Sharings for Input and Output Gates:** For each group of  $k$  input gates or output gates, let  $\alpha$  denote the output wires of these  $k$  input gates or the input wires of these  $k$  output gates. Recall that all parties obtain  $[\mathbf{o}]_{n-1}$  and  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, ([c]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$  in  $\mathcal{F}_{\text{PrepIndMal}}$ . Also recall that all parties hold  $\{[\lambda_{\alpha_i} \cdot \mathbf{1}]_{n-k}, \langle \Delta \cdot \lambda_{\alpha_i} \rangle\}_{i=1}^k$ . All parties locally compute  $[\lambda_\alpha]_{n-1} = \left( \sum_{i=1}^k e_i * [\lambda_{\alpha_i} \cdot \mathbf{1}]_{n-k} \right) + [\mathbf{o}]_{n-1}$ .

<sup>a</sup> We will discuss how parties locally refresh an additive sharing in the full version.

**Lemma 1.** *Protocol  $\Pi_{\text{PrepMal}}$  securely computes  $\mathcal{F}_{\text{PrepMal}}$  in the  $\mathcal{F}_{\text{PrepIndMal}}$ -hybrid model against a malicious adversary who controls  $t$  out of  $n$  parties.*

Lemma 1 is proven in the full version of this paper.

**Communication Complexity of  $\Pi_{\text{PrepMal}}$ .** The only communication in Protocol  $\Pi_{\text{PrepMal}}$  (ignoring calls to  $\Pi_{\text{PrepIndMal}}$ ) happens in Step 4a. This amounts to  $2(n - 1)$  shares sent to  $P_1$ , per group of  $k$  multiplication gates, so  $\frac{2n-2}{k} = \frac{4n-4}{\epsilon \cdot n+2} \leq \frac{4}{\epsilon}$  per multiplication gate.

## 6 Circuit-Independent Preprocessing Phase

In this section, we discuss how to realize the ideal functionality  $\mathcal{F}_{\text{PrepIndMal}}$  for the circuit-independent preprocessing phase. Recall that  $k = (n - t + 1)/2$ . For simplicity, we only focus on the scenario where  $t \geq n/2$ . Due to space constraints, part of the procedures we use to instantiate  $\mathcal{F}_{\text{PrepIndMal}}$  appear in the full version of this paper. Here, we focus on the fundamental aspects of the instantiation. Recall that  $\mathcal{F}_{\text{PrepIndMal}}$  is in charge of generating the following correlations:

1. The global random key  $[\Delta|_1]_t, \dots, [\Delta|_k]_t$ ;
2. Shamir sharings  $[\lambda_\alpha \cdot \mathbf{1}]_{n-k}$  and additive sharings  $\langle \Delta \cdot \lambda_\alpha \rangle$  for every wire  $\alpha$ ;
3. A tuple  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, ([\mathbf{c}]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$  and shares of zero  $[\mathbf{o}^{(1)}]_{n-1}, [\mathbf{o}^{(2)}]_{n-1}, [\mathbf{o}^{(3)}]_{n-1}$  for every group of  $k$  multiplication gates;
4. A tuple  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, ([\mathbf{c}]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$  and a share of zero  $[\mathbf{o}]_{n-1}$  for every group of  $k$  input or output gates.

In this section we will focus our attention on how to generate the packed Beaver triples  $([\mathbf{a}]_{n-k}, [\mathbf{b}]_{n-k}, ([\mathbf{c}]_{n-1}, \{\langle \Delta \cdot c_i \rangle\}_{i=1}^k))$ . All of the remaining correlations are discussed in full detail in the full version of this paper.

**Building Blocks: OLE and VOLE.** It is known that protocols in the dishonest majority setting require computational assumptions. In our work, these appear in the use of oblivious linear evaluation. Here, we make use of two functionalities,  $\mathcal{F}_{\text{nVOLE}}$  and  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$ , which sample OLE correlations as follows. We consider an expansion function  $\text{Expand} : S \rightarrow \mathbb{F}_p^m$  with seed space  $S$  and output length  $m$ , ultimately corresponding to the amount of correlations we aim at generating.

- $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  is a two-party functionality such that, on input seeds  $s_a$  from party  $P_A$  and  $s_b$  from party  $P_B$ , samples  $\mathbf{v} \leftarrow \mathbb{F}_p^m$ , and outputs  $\mathbf{w} = \mathbf{u} * \mathbf{x} - \mathbf{v}$  to  $P_A$  and  $\mathbf{v}$  to  $P_B$ . Here,  $\mathbf{u} = \text{Expand}(s_a)$  and  $\mathbf{v} = \text{Expand}(s_b)$ . Notice that in this functionality the parties can choose their inputs (at least, choose their seeds).
- $\mathcal{F}_{\text{nVOLE}}$  is an  $n$ -party functionality that first distributes  $\Delta^i \leftarrow \mathbb{F}$  to each party  $P_i$  in an initialize phase, and then, to sample  $m$  correlations, the functionality sends  $s^i, (\mathbf{w}_j^i, \mathbf{v}_j^i)_{j \neq i}$  to each party  $P_i$ , where  $s^i$  is a uniformly random seed,  $\mathbf{v}_j^i \leftarrow \mathbb{F}_p^m$ , and  $\mathbf{w}_j^i = \mathbf{u}^i \cdot \Delta^j - \mathbf{v}_j^i$ , and  $\mathbf{u}^i = \text{Expand}(s^i)$ . Notice that in this functionality, the parties do not choose their inputs (seeds), but rather, the functionality samples the seeds and sends them to the parties.

The functionalities above are presented in full detail in the full version of this paper. At a high level,  $\mathcal{F}_{\text{nVOLE}}$  is used to generate authenticated sharings of a uniformly random value, and  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$ , which allows the parties to set their inputs, is used to secure multiply two already-shared secret values.  $\mathcal{F}_{\text{nVOLE}}$  can be instantiated using pseudo-random correlator generators, as suggested in [25]. On the other hand, for  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  we can use the implementation from [25]. As we are using *exactly* the same functionalities as in [25], we refer the reader to that work for instantiations and complexity measures.

**Omitted Procedures.** For our triple generation protocol we will make use of a series of procedures that are described in full detail in the full version of this paper. These procedures are the following:

- $\pi_{\text{RandSh}}$ : this procedure generates sharings (under some secret-sharing scheme, which will be clear from context) of uniformly random values. For this, the trick of using a Vandermonde matrix for randomness extraction from [13] is used.
- $\pi_{\text{DegReduce}}$ : this procedure takes as input a sharing  $[\mathbf{u}]_{n-1}$  and outputs  $[\mathbf{u}]_{n-k}$ . This is achieved by using the trick of masking with a random value  $[\mathbf{r}]_{n-1}$ , opening, and unmasking with  $[\mathbf{r}]_{n-k}$ . This random pair is generated using  $\pi_{\text{RandSh}}$ .
- $\pi_{\text{AddTran}}$ : this procedure takes as input sharings  $(\langle \Delta \cdot u_1 \rangle, \dots, \langle \Delta \cdot u_k \rangle)$  and converts them to  $[\Delta \cdot \mathbf{u}]_{n-k}$ . Once again, the trick of masking with a random sharing  $\langle r_1 \rangle, \dots, \langle r_k \rangle$ , opening, and unmasking with  $[\mathbf{r}]$ , is used. The sharing  $[\mathbf{r}]$  is obtained using  $\pi_{\text{RandSh}}$ , and each  $\langle r_i \rangle$  can be derived from it non-interactively.
- $\pi_{\text{MACKey}}$ : this procedure enables the parties to obtain individual Shamir sharings of the global MAC key  $[\Delta|_1]_t, \dots, [\Delta|_k]_t$ , starting from additive shares of it  $\langle \Delta \rangle$  which are obtained using  $\mathcal{F}_{\text{nVOLE}}$ . This is done by using the standard trick of masking with a random value  $\langle r \rangle$ , opening, and unmasking with each  $[r|_i]_t$ . These random sharings are obtained using  $\pi_{\text{RandSh}}$ .
- $\pi_{\text{Auth}}$ : this procedure takes as input sharings  $(\langle u_1 \rangle, \dots, \langle u_k \rangle)$  to  $([\mathbf{u}]_{n-1}, \{\langle \Delta \cdot u_i \rangle\}_{i=1}^k)$ . The trick here is to mask with  $\langle r_1 \rangle, \dots, \langle r_k \rangle$ , open, and adding  $[\mathbf{r}]_{n-1}$  to obtain  $[\mathbf{u}]_{n-1}$ . The authenticated part can be obtained by first multiplying locally by each  $[\Delta|_i]_t$  and then adding each  $\langle \Delta \cdot r_i \rangle$ . The pair  $([\mathbf{r}]_{n-1}, \{\langle \Delta \cdot r_i \rangle\}_{i=1}^k)$  is produced using  $\pi_{\text{RandSh}}$ .

**Preparing Packed Beaver Triples with Authentications.** The procedure to generate packed Beaver triples with authentications,  $\pi_{\text{Triple}}$ , is described below. This protocol calls  $\pi_{\text{DegReduce}}$  twice,  $\pi_{\text{AddTran}}$  twice, and  $\pi_{\text{Auth}}$  once per triple.

**Procedure 4:**  $\pi_{\text{Triple}}$ 

**Initialization:** All parties run the following initialization step only once.

1. Each  $P_i$  calls  $\mathcal{F}_{\text{nVOLE}}$  with input **Init** and receives  $\Delta^i$ .
2. All parties invoke  $\pi_{\text{RandSh}}$  to prepare random sharings  $\{[r]_i\}_{i=1}^k$  and then invoke  $\pi_{\text{MACKey}}$  and obtain  $\{[\Delta]_i\}_{i=1}^k$ .

**Generation:**

1. Each  $P_i$  calls  $\mathcal{F}_{\text{nVOLE}}$  twice with input **Extend** and receives the seeds  $s_a^i, s_b^i$ . Use the outputs to define degree- $(n-1)$  packed Shamir sharings  $\{[a_\ell]_{n-1}\}_{\ell=1}^m, \{[b_\ell]_{n-1}\}_{\ell=1}^m$ , where  $m$  is the output length of the expansion function defined in  $\mathcal{F}_{\text{nVOLE}}$ , such that the  $i$ -th shares of  $\{[a_\ell]_{n-1}\}_{\ell=1}^m$  are  $\text{Expand}(s_a^i)$ , and the  $i$ -th shares of  $\{[b_\ell]_{n-1}\}_{\ell=1}^m$  are  $\text{Expand}(s_b^i)$ . All parties locally compute and refresh  $\{(\langle \Delta \cdot a_{\ell,1} \rangle, \dots, \langle \Delta \cdot a_{\ell,k} \rangle)\}_{\ell=1}^m$  and  $\{(\langle \Delta \cdot b_{\ell,1} \rangle, \dots, \langle \Delta \cdot b_{\ell,k} \rangle)\}_{\ell=1}^m$ .
2. Every ordered pair  $(P_i, P_j)$  calls  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  with  $P_i$  sending  $s_a^i$  and  $P_j$  sending  $s_b^j$ .  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  sends back  $\mathbf{u}^{i,j}$  to  $P_i$  and  $\mathbf{v}^{j,i}$  to  $P_j$  such that  $\mathbf{u}^{i,j} + \mathbf{v}^{j,i} = \text{Expand}(s_a^i) * \text{Expand}(s_b^j)$ . All parties locally compute  $\{(\langle c_{\ell,1} \rangle, \dots, \langle c_{\ell,k} \rangle)\}_{\ell=1}^m$  where  $\mathbf{c}_\ell = \mathbf{a}_\ell * \mathbf{b}_\ell$ .
3. All parties invoke  $\pi_{\text{RandSh}}$  to prepare  $m$  random sharings in the form of  $([r]_{n-k}, [r]_{n-1})$ . For all  $\ell \in \{1, \dots, m\}$ , consume a pair of random sharings  $([r]_{n-k}, [r]_{n-1})$  and invoke  $\pi_{\text{DegReduce}}$  to transform  $[a_\ell]_{n-1}$  to  $[a_\ell]_{n-k}$ . Repeat this step for  $\{[b_\ell]_{n-1}\}_{\ell=1}^m$ .
4. All parties invoke  $\pi_{\text{RandSh}}$  to prepare  $m$  random sharings in the form of  $[r]_{n-k}$ . For all  $\ell \in \{1, \dots, m\}$ , consume a random sharing  $[r]_{n-k}$  and invoke  $\pi_{\text{AddTran}}$  to transform  $(\langle \Delta \cdot a_{\ell,1} \rangle, \dots, \langle \Delta \cdot a_{\ell,k} \rangle)$  to  $[\Delta \cdot \mathbf{a}_\ell]_{n-k}$ . Repeat this step for  $\{(\langle \Delta \cdot b_{\ell,1} \rangle, \dots, \langle \Delta \cdot b_{\ell,k} \rangle)\}_{\ell=1}^m$ .
5. All parties follow Step 1 to prepare  $m$  random sharings with authentications in the form of  $([r]_{n-1}, \{\langle \Delta \cdot r_i \rangle\}_{i=1}^k)$ . For all  $\ell \in \{1, \dots, m\}$ , consume a random sharing  $([r]_{n-1}, \{\langle \Delta \cdot r_i \rangle\}_{i=1}^k)$  and invoke  $\pi_{\text{Auth}}$  to transform  $(\langle c_{\ell,1} \rangle, \dots, \langle c_{\ell,k} \rangle)$  to  $([c_\ell]_{n-1}, \{\langle \Delta \cdot c_{\ell,i} \rangle\}_{i=1}^k)$ .

We remark that the triples produced by  $\pi_{\text{Triple}}$  may not be correct, but this can be checked by running a verification step in which the parties generate an extra triple and “sacrifice” it in order to check for correctness. This is described in the full version, where three procedures  $\pi_{\text{Sacrifice}}$ ,  $\pi_{\text{CheckZero}}$  and  $\pi_{\text{VerifyDeg}}$  to perform this check are introduced.

*Communication Complexity of  $\pi_{\text{Triple}}$ .* This is derived as follows

- (Step 3) Two calls to  $\pi_{\text{RandSh}}$  to generate two pairs  $([r]_{n-k}, [r]_{n-1})$ , which costs  $2n$ , and two calls to  $\pi_{\text{DegReduce}}$ , which costs  $2(2n-k)$ . These sum up to  $6n-2k$
- (Step 4) Two calls to  $\pi_{\text{RandSh}}$  to generate  $[r]_{n-k}$  and two calls to  $\pi_{\text{AddTran}}$ . These add up to  $2(n/2) + 2(k \cdot (n-2) + n + 1)$ .
- (Step 5) One call to  $\pi_{\text{Auth}}$ , which is  $k \cdot (n-2) + n + 1$ .

The above totals  $k \cdot (3n-8) + 10n + 3$ .

*Remark 1 (On the output size of  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  and  $\mathcal{F}_{\text{nVOLE}}$ ).* We make the crucial observation that, in order to obtain  $m$  packed multiplication triples, we require the `Expand` function used in Functionalities  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  and  $\mathcal{F}_{\text{nVOLE}}$  to output  $m$  field elements. However, since each such packed triple is used for a group of  $k$  multiplication gates, this effectively means that, if there are  $|C|$  multiplication gates in total, we only require `Expand` to output  $|C|/k \approx 2|C|/(\epsilon n)$  correlations. In contrast, as we will see in the full version of this paper, the best prior work `Turbospeedz` [3], when instantiated with the preprocessing from Le Mans [25], would require  $|C|$  correlations from the  $\mathcal{F}_{\text{nVOLE}}$  and  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$ . As a result, we manage to reduce by a factor of  $k$  the expansion requirements on `VOLE/OLE` techniques, which has a direct effect on the resulting efficiency since this allows us to choose better parameters for the realizations of  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  and  $\mathcal{F}_{\text{nVOLE}}$ . We do not explore these concrete effects in efficiency as it goes beyond the scope of our work, but we refer the reader to [25] where an instantiation of  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  and a discussion on `PCG-based  $\mathcal{F}_{\text{nVOLE}}$`  is presented.

**Final Circuit-Independent Preprocessing Protocol.** In the full version of this paper, we present the final protocol,  $\Pi_{\text{PrepIndMal}}$ , that puts together the pieces we have discussed so far, together with the techniques to generate the remaining correlations, in order to instantiate Functionality  $\mathcal{F}_{\text{PrepIndMal}}$ . The proof of the lemma below will be available in the full version. We also analyze the communication complexity of  $\Pi_{\text{PrepIndMal}}$  and conclude that, per multiplication gate (ignoring terms that are independent of the circuit size),  $6n + \frac{35}{\epsilon}$  elements are required.

**Lemma 2.** *Protocol  $\Pi_{\text{PrepIndMal}}$  securely computes  $\mathcal{F}_{\text{PrepIndMal}}$  in the  $\{\mathcal{F}_{\text{OLE}}^{\text{prog}}, \mathcal{F}_{\text{nVOLE}}, \mathcal{F}_{\text{Commit}}, \mathcal{F}_{\text{Coin}}\}$ -hybrid model against a malicious adversary who controls  $t$  out of  $n$  parties.*

## 7 Implementation and Experimental Results

We have fully implemented the three phases of `SUPERPACK`,  $\Pi_{\text{Online}}$ ,  $\Pi_{\text{PrepMal}}$  and  $\Pi_{\text{PrepIndMal}}$ , only ignoring the calls to the  $\mathcal{F}_{\text{OLE}}^{\text{prog}}$  and  $\mathcal{F}_{\text{nVOLE}}$  functionalities for the implementation of  $\Pi_{\text{PrepIndMal}}$ . In this section we discuss our experimental results.

*Implementation Setup.* We implement `SUPERPACK` by using as a baseline the code of `TURBOPACK` [15].<sup>5</sup> As `TURBOPACK`, our program is written in `C++` with no dependencies beyond the standard library. Our implementation includes fully functional networking code. However, for the experiments, we deploy the protocol as multiple processes in a single machine, and emulate real network conditions using the package `netem`<sup>7</sup>, which allows us to set bandwidth and

<sup>5</sup> `TURBOPACK` is available at <https://github.com/deescuderoo/turbopack>.

<sup>6</sup> `SUPERPACK` is available at <https://github.com/ckweng/SuperPack>.

<sup>7</sup> <https://wiki.linuxfoundation.org/networking/netem>.

**Table 2.** Running times in seconds of SUPERPACK across its three different phases, for different circuit widths, number of parties, and values of  $\epsilon$ . Each cell is a triple corresponding to the runtimes of the online phase, circuit-dependent offline phase, and circuit-independent offline phase (ignoring OLE calls), respectively. All the circuits have depth 10.

Width	# Parties	Percentage of corrupt parties			
		90%	80%	70%	60%
100	16	0.63, 0.07, 1.22	0.56, 0.06, 1.26	0.33, 0.06, 1.25	0.34, 0.06, 1.26
	32	0.47, 0.11, 2.86	0.47, 0.11, 2.90	0.75, 0.11, 2.86	0.62, 0.09, 2.87
	48	0.35, 0.18, 5.77	0.63, 0.16, 6.41	0.68, 0.15, 6.33	0.68, 0.13, 6.01
1k	16	0.44, 0.21, 2.10	0.45, 0.16, 2.14	0.45, 0.20, 2.1	0.66, 0.16, 2.15
	32	0.50, 0.69, 8.38	0.61, 0.63, 9.08	0.58, 0.54, 9.05	0.64, 0.62, 8.78
	48	0.59, 1.46, 21.31	0.97, 1.15, 25.93	0.90, 1.05, 24.70	0.69, 1.01, 24.20
10k	16	1.74, 2.03, 14.10	1.49, 1.64, 13.36	1.45, 1.64, 13.39	1.28, 1.43, 12.44
	32	2.36, 6.25, 70.71	2.03, 5.60, 73.98	2.26, 4.80, 70.47	2.32, 4.45, 67.16
	48	3.24, 12.48, 196.97	3.19, 10.39, 238.32	3.49, 9.49, 227.80	4.14, 7.87, 201.17
100k	16	11.84, 15.39, 147.03	9.60, 12.46, 140.01	9.63, 12.56, 140.18	8.74, 10.68, 129.89
	32	19.84, 64.61, 714.02	17.46, 46.56, 749.22	18.39, 38.70, 716.26	19.18, 35.03, 682.54
	48	27.62, 124.22, 1978.42	27.56, 103.55, 2374.39	31.55, 92.74, 2256.70	36.98, 78.55, 1998.26

latency constraints. We use the same machine as in [15] for the experiments, namely an AWS `c5.metal` instance with 96 vCPUs and 192 GiB of memory. For our protocol, we use a finite field  $\mathbb{F} = \mathbb{F}_p$  where  $p = 2^{61} - 1$ . We explore how the performance of our protocol is affected by the parameters including the number of parties  $n$ , the width and depth of the circuit, the network bandwidth and the values of  $\epsilon$  such that  $t = n(1 - \epsilon)$  is the threshold for corrupted parties.

*End-to-End Runtimes.* We first report the running times of our SUPERPACK protocol for each of the three phases: circuit-independent preprocessing, circuit-dependent preprocessing, and online phase. The results are given in Table 2. In our experiments, we show the running time of our protocol for different parameters. We throttle the bandwidth to 1Gbps and network latency to 1ms to simulate a LAN setting. We generate four generic 10-layer circuits of widths 100, 1k, 10k and 100k. For each circuit, we benchmark the SUPERPACK protocol of which the number of parties are chosen from  $\{16, 32, 48\}$ . After fixing the circuit and parties, the percentage of corrupt parties varies from 60%, 70%, 80% and 90%. Generally the running time increases as the width and number of parties increase. As demonstrated in Table 2, the majority of running time is incurred by the circuit-independent preprocessing. For  $n = 48$  and width larger than 1k, the online phase only occupies less than 5% of the total running time. Furthermore, it is important to observe that the runtimes of the online and circuit-dependent offline phases do not grow at the same rate as the runtimes for the circuit-independent offline phase. This is consistent with what we expect: as can be seen from Table 1, the communication in the first two phases is independent of the number of parties for a given  $\epsilon$ , which is reflected in the low increase rate in runtimes for these phases (there is still a small but noticeable growth, but

this is not surprising since even though communication is constant, *computation* is not). In contrast, the communication in the circuit-independent offline phase depends linearly on the number of parties, which impacts runtimes accordingly.

*Experimental Comparison to Turbospeedz.* Now we compare the online phase of our protocol and compare it against that of Turbospeedz [3],<sup>8</sup> for a varying number of parties  $n$  and parameter  $\epsilon$ . We fix the circuit to have width 100k and depth 10, but we vary the bandwidth in  $\{500, 100, 50, 10\}$ mbps. The results are given in Table 3. Notice that we report the improvement factor of our online phase with respect to that of Turbospeedz. The concrete runtimes will be available in the full version. We also report the communication factors between our protocol and Turbospeedz, for reference.

Table 3 shows interesting patterns. First, as expected (and as analyzed theoretically in the full version), our improvement factor with respect to Turbospeedz improves (*i.e.* increases) as the number of parties grows—since in this case communication in Turbospeedz grows but in our case remains constant—or as the percentage of corruptions decreases—since in this case we can pack more secrets per sharing. Now, notice the following interesting behavior. The last rows next to the “comm. factor” rows represent the improvement factor of our online phase with respect to Turbospeedz, in terms of *communication*. In principle, this is the improvement factor we would expect to see in terms of runtimes. However, we observe that the expected factor is only reasonably close to the experimental ones for low bandwidths such as 10, 50 and 100 mbps. For the larger bandwidth of 500 mbps, we see that the experimental improvement factors are much lower than the ones we would expect, and in fact, there are several cases where we expect our protocol to be even slightly better, and instead it performs *worse*.

The behavior above can be explained in different ways. First, we notice that it is not surprising that our improvement factor increases as the bandwidth decreases, since in this case the execution of the protocol becomes communication bounded, and computation overhead becomes negligible. In contrast, when the bandwidth is high, communication no longer becomes a bottleneck, and computation plays a major role. Here is where our protocol is in a slight disadvantage: in SUPERPACK, the parties (in particular  $P_1$ ) must perform polynomial interpolation in a regular basis, while in Turbospeedz these operations correspond to simple field element multiplications, which are less expensive. We remark that our polynomial interpolation is very rudimentary, and a more optimized implementation (*e.g.* using FFTs) may be the key to bridging the gap between our protocol and Turbospeedz, even for the case when bandwidth is large. Finally, we remark that SUPERPACK remains the best option even with high bandwidth when the fraction of honest parties is large enough.

---

<sup>8</sup> We implemented the online phase of Turbospeedz in our framework for a fair comparison.

**Table 3.** Improvement factors of our online protocol with respect to the online phase in Turbospeedz, for a varying number of parties,  $\epsilon$  and network bandwidth. The network delay is 1ms for the simulation of LAN network. The number represents how much better (or worse) our online phase is with respect to that of Turbospeedz. The circuits have depth 10 and width 10k. In the final five rows we show the corresponding factors but measuring communication complexity, instead of runtimes.

Bandwidth	# Parties	Percentage of corrupt parties			
		90%	80%	70%	60%
500 mbps	<b>16</b>	0.51	0.44	0.42	0.50
	<b>32</b>	0.55	0.68	0.68	0.72
	<b>48</b>	0.58	0.87	1.00	1.14
	<b>64</b>	0.75	0.92	1.30	1.22
	<b>80</b>	0.95	1.27	1.57	1.40
100 mbps	<b>16</b>	0.97	1.08	1.05	1.20
	<b>32</b>	1.43	1.67	1.88	1.95
	<b>48</b>	1.51	2.38	2.78	3.07
	<b>64</b>	2.08	2.95	3.37	3.47
	<b>80</b>	2.51	3.88	4.57	4.56
50 mbps	<b>16</b>	1.08	1.31	1.31	1.45
	<b>32</b>	1.57	1.99	2.43	2.44
	<b>48</b>	1.73	2.88	3.43	3.76
	<b>64</b>	2.24	3.60	4.55	4.34
	<b>80</b>	2.76	4.51	5.30	5.59
10 mbps	<b>16</b>	1.10	1.40	1.39	1.53
	<b>32</b>	1.58	2.00	2.53	2.68
	<b>48</b>	1.81	3.04	3.61	3.94
	<b>64</b>	2.31	3.60	4.73	5.28
	<b>80</b>	2.91	4.56	5.73	6.22
<b>Comm. factor</b>	<b>16</b>	0.48	0.85	1.12	1.28
	<b>32</b>	0.96	1.71	2.24	2.56
	<b>48</b>	1.44	2.56	3.36	3.84
	<b>64</b>	1.92	3.41	4.48	5.12
	<b>80</b>	2.4	4.27	5.6	6.4

**Acknowledgments.** This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates (“JP Morgan”), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument,



financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful. 2022 JP Morgan Chase & Co. All rights reserved.

## References

1. Beaver, D.: Efficient multiparty protocols using circuit randomization, pp. 420–432 (1992). [https://doi.org/10.1007/3-540-46766-1\\_34](https://doi.org/10.1007/3-540-46766-1_34)
2. Beck, G., Goel, A., Jain, A., Kaptchuk, G.: Order-C secure multiparty computation for highly repetitive circuits, pp. 663–693 (2021). [https://doi.org/10.1007/978-3-030-77886-6\\_23](https://doi.org/10.1007/978-3-030-77886-6_23)
3. Ben-Efraim, A., Nielsen, M., Omri, E.: Turbospeedz: double your online SPDZ! Improving SPDZ using function dependent preprocessing, pp. 530–549 (2019). [https://doi.org/10.1007/978-3-030-21568-2\\_26](https://doi.org/10.1007/978-3-030-21568-2_26)
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract), pp. 1–10 (1988). <https://doi.org/10.1145/62212.62213>
5. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation, pp. 169–188 (2011). [https://doi.org/10.1007/978-3-642-20465-4\\_11](https://doi.org/10.1007/978-3-642-20465-4_11)
6. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear PCPs, pp. 67–97 (2019). [https://doi.org/10.1007/978-3-030-26954-8\\_3](https://doi.org/10.1007/978-3-030-26954-8_3)
7. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Efficient fully secure computation via distributed zero-knowledge proofs, pp. 244–276 (2020). [https://doi.org/10.1007/978-3-030-64840-4\\_9](https://doi.org/10.1007/978-3-030-64840-4_9)
8. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols, pp. 136–145 (2001). <https://doi.org/10.1109/SFCS.2001.959888>
9. Chida, K., et al.: Fast large-scale honest-majority MPC for malicious adversaries, pp. 34–64 (2018). [https://doi.org/10.1007/978-3-319-96878-0\\_2](https://doi.org/10.1007/978-3-319-96878-0_2)
10. Couteau, G.: A note on the communication complexity of multiparty computation in the correlated randomness model, pp. 473–503 (2019). [https://doi.org/10.1007/978-3-030-17656-3\\_17](https://doi.org/10.1007/978-3-030-17656-3_17)
11. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography, pp. 445–465 (2010). [https://doi.org/10.1007/978-3-642-13190-5\\_23](https://doi.org/10.1007/978-3-642-13190-5_23)
12. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits, pp. 1–18 (2013). [https://doi.org/10.1007/978-3-642-40203-6\\_1](https://doi.org/10.1007/978-3-642-40203-6_1)
13. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation, pp. 572–590 (2007). [https://doi.org/10.1007/978-3-540-74143-5\\_32](https://doi.org/10.1007/978-3-540-74143-5_32)
14. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption, pp. 643–662 (2012). [https://doi.org/10.1007/978-3-642-32009-5\\_38](https://doi.org/10.1007/978-3-642-32009-5_38)
15. Escudero, D., Goyal, V., Polychroniadou, A., Song, Y.: TurboPack: honest majority MPC with constant online communication, pp. 951–964 (2022). <https://doi.org/10.1145/3548606.3560633>
16. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract), pp. 699–710 (1992). <https://doi.org/10.1145/129712.129780>

17. Genkin, D., Ishai, Y., Polychroniadou, A.: Efficient multi-party computation: from passive to active security via secure SIMD circuits, pp. 721–741 (2015). [https://doi.org/10.1007/978-3-662-48000-7\\_35](https://doi.org/10.1007/978-3-662-48000-7_35)
18. Genkin, D., Ishai, Y., Prabhakaran, M.M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing, pp. 495–504. STOC 2014, ACM, New York, NY, USA (2014). <https://doi.org/10.1145/2591796.2591861>
19. Goldwasser, S., Lindell, Y.: Secure multi-party computation without agreement. *J. Cryptol.* **18**(3), 247–287 (2005). <https://doi.org/10.1007/s00145-005-0319-z>
20. Goyal, V., Li, H., Ostrovsky, R., Polychroniadou, A., Song, Y.: ATLAS: efficient and scalable MPC in the honest majority setting, pp. 244–274 (2021). [https://doi.org/10.1007/978-3-030-84245-1\\_9](https://doi.org/10.1007/978-3-030-84245-1_9)
21. Goyal, V., Polychroniadou, A., Song, Y.: Unconditional communication-efficient MPC via hall’s marriage theorem, pp. 275–304 (2021). [https://doi.org/10.1007/978-3-030-84245-1\\_10](https://doi.org/10.1007/978-3-030-84245-1_10)
22. Goyal, V., Polychroniadou, A., Song, Y.: Sharing transformation and dishonest majority MPC with packed secret sharing, pp. 3–32 (2022). [https://doi.org/10.1007/978-3-031-15985-5\\_1](https://doi.org/10.1007/978-3-031-15985-5_1)
23. Goyal, V., Song, Y.: Malicious security comes free in honest-majority MPC. Cryptology ePrint Archive, Report 2020/134 (2020). <https://eprint.iacr.org/2020/134>
24. Lindell, Y., Nof, A.: A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority, pp. 259–276 (2017). <https://doi.org/10.1145/3133956.3133999>
25. Rachuri, R., Scholl, P.: Le mans: Dynamic and fluid MPC for dishonest majority, pp. 719–749 (2022). [https://doi.org/10.1007/978-3-031-15802-5\\_25](https://doi.org/10.1007/978-3-031-15802-5_25)
26. Shamir, A.: How to share a secret. *Commun. ACM* **22**(11), 612–613 (1979). <https://doi.org/10.1145/359168.359176>



# Detect, Pack and Batch: Perfectly-Secure MPC with Linear Communication and Constant Expected Time

Ittai Abraham<sup>1</sup>, Gilad Asharov<sup>2</sup>, Shravani Patil<sup>3</sup>(✉), and Arpita Patra<sup>3</sup>

<sup>1</sup> VMware Research, 5 Sapir St., 4685209 Herzliya, Israel  
iabraham@vmware.com

<sup>2</sup> Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel  
Gilad.Asharov@biu.ac.il

<sup>3</sup> Indian Institute of Science, Bangalore, India  
{shravanip,arpita}@iisc.ac.in

**Abstract.** We prove that perfectly-secure optimally-resilient secure Multi-Party Computation (MPC) for a circuit with  $C$  gates and depth  $D$  can be obtained in  $\mathcal{O}((Cn+n^4+Dn^2)\log n)$  communication complexity and  $\mathcal{O}(D)$  expected time. For  $D \ll n$  and  $C \geq n^3$ , this is the **first** perfectly-secure optimal-resilient MPC protocol with **linear** communication complexity per gate and **constant** expected time complexity per layer.

Compared to state-of-the-art MPC protocols in the player elimination framework [Beerliova and Hirt TCC'08, and Goyal, Liu, and Song CRYPTO'19], for  $C > n^3$  and  $D \ll n$ , our results significantly improve the run time from  $\Theta(n+D)$  to expected  $\mathcal{O}(D)$  while keeping communication complexity at  $\mathcal{O}(Cn \log n)$ .

Compared to state-of-the-art MPC protocols that obtain an expected  $\mathcal{O}(D)$  time complexity [Abraham, Asharov, and Yanai TCC'21], for  $C > n^3$ , our results significantly improve the communication complexity from  $\mathcal{O}(Cn^4 \log n)$  to  $\mathcal{O}(Cn \log n)$  while keeping the expected run time at  $\mathcal{O}(D)$ .

One salient part of our technical contribution is centered around a new primitive we call *detectable secret sharing*. It is perfectly-hiding, weakly-binding, and has the property that either reconstruction succeeds, or  $\mathcal{O}(n)$  parties are (privately) detected. On the one hand, we show that detectable secret sharing is sufficiently powerful to generate multiplication triplets needed for MPC. On the other hand, we show how to share  $p$  secrets via detectable secret sharing with communication complexity of just  $\mathcal{O}(n^4 \log n + p \log n)$ . When sharing  $p \geq n^4$  secrets, the communication cost is amortized to just  $\mathcal{O}(1)$  per secret.

Our second technical contribution is a new Verifiable Secret Sharing protocol that can share  $p$  secrets at just  $\mathcal{O}(n^4 \log n + pn \log n)$  word complexity. When sharing  $p \geq n^3$  secrets, the communication cost is amortized to just  $\mathcal{O}(n)$  per secret. The best prior required  $\mathcal{O}(n^3)$  communication per secret.

## 1 Introduction

In the setting of secure multiparty computation (MPC),  $n$  distrustful parties jointly compute a function on their inputs while keeping their inputs private. Security should be preserved even in the presence of an external entity that controls some parties and coordinates their behavior. We consider in this paper the most demanding setting: perfect security with optimal resilience. Perfect security means that the adversary is all-powerful and that the protocol has zero probability of error. Optimal resilience means that the number of corruptions is at most  $t < n/3$ . Such protocols come with desirable properties: They guarantee adaptive security (with some caveats [5, 17]) and remain secure under universal composition [34].

The seminal protocols of Ben-Or, Goldwasser, and Wigderson [13], and Chaum, Crépeau and Damgård [19] led the foundations of this setting. Since then, there are, in general, two families of protocols:

1. **Efficient but slow:** These protocols [12, 30, 32] ([12] test-of-time award) have  $\mathcal{O}(n \log n)$  communication complexity per multiplication gate. Still, the running time of these protocols is at least  $\Theta(n)$  rounds, even if the depth of the circuit is much smaller  $D \ll n$ . Specifically:

**Theorem 1.1.** *For an arithmetic circuit with  $C$  multiplication gates and depth  $D$  there exists a perfectly-secure, optimally-resilient MPC protocol with  $\mathcal{O}(n^5 \log n + Cn \log n)$  bits communication complexity and  $\Omega(n + D)$  expected number of rounds.*

The protocol requires  $\mathcal{O}(n^3 \log n + Cn \log n)$  bits of point-to-point communication and  $n$  sequential invocations of broadcast of  $\mathcal{O}(\log n)$  bits each, with  $\Omega(n + D)$  rounds. Using the broadcast implementation of [1], this becomes the complexity of Theorem 1.1. Alternatively, using the implementation of [15, 23], the protocol can be more efficient, but even more slower:  $\mathcal{O}(n^3 \log n + Cn \log n)$  bits communication complexity and  $\Omega(n^2 + D)$  number of rounds.

2. **Fast but not efficient:** This line of protocols [2, 7, 13, 19, 24, 29] run at  $\mathcal{O}(D)$  expected number of rounds, but require  $\Omega(n^4 \log n)$  communication complexity per multiplication gate.

**Theorem 1.2.** *For an arithmetic circuit with  $C$  multiplication gates and depth  $D$  there exists a perfectly-secure, optimally-resilient MPC protocol with  $\Omega(Cn^4 \log n)$  communication complexity and  $\mathcal{O}(D)$  expected number of rounds.*

In the broadcast hybrid model, the protocol requires  $\mathcal{O}(n^3 \log n)$  bits of communication complexity over point-to-point channels and  $\mathcal{O}(n^3 \log n)$  bits broadcast, in  $\mathcal{O}(D)$  number of rounds. Theorem 1.2 reports the communication complexity using the broadcast implementation of [1]. Using [15, 23] for implementing the broadcast, the number of rounds is increased to  $\Omega(n + D)$ .

## Our Main Result

Our main result is that the best of both families is possible to achieve simultaneously. For the first time, we provide a perfectly-secure, optimally-resilient MPC protocol that has **both**  $\mathcal{O}(n \log n)$  communication complexity per multiplication gate and  $\mathcal{O}(D)$  expected time complexity.

**Theorem 1.3** (Main Result). *For a circuit with  $C$  multiplication gates and depth  $D$  there exists a perfectly-secure, optimally-resilient MPC protocol with  $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$  communication complexity and  $\mathcal{O}(D)$  expected number of rounds.*

In the broadcast-hybrid model, the total communication complexity over point-to-point is  $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$ , and each party has to broadcast at most  $\mathcal{O}(n^2 \log n)$  bits. Using [1] for implementing the broadcast, we obtain Theorem 1.3. Compared to [12,30], for  $D \ll n$ , our result provides up to an  $\mathcal{O}(n)$  improvement in round complexity while keeping the same linear communication complexity (and also improving the communication complexity for  $C \in o(n^4)$ ). Compared to [2], for  $C > n^3$ , our result provides an  $\mathcal{O}(n^3)$  improvement(!) in the communication complexity while keeping the same  $\mathcal{O}(D)$  expected round complexity.

We remark that in many practical settings, a large set of parties may want to compute a shallow depth circuit in a robust manner. For instance, consider a network with 200ms latency and channels of 1Gbps, and consider a highly parallel circuit with 1M gates, depth  $D = 10$ , and  $n = 200$  parties. Then, the round complexity of our protocol is  $\mathcal{O}(D)$ , which results in a delay of  $10 \cdot 200\text{ms} = 2$  s. The delay associated with the communication complexity is smaller: each party sends or receives  $(C + Dn + n^3) \log n$  bits, which over 1Gbps channel results in a delay of 0.08 s. In [30], the delay due to the round complexity is  $\mathcal{O}(n + D)$ , which results in a delay of  $210 \cdot 200\text{ms} = 42$  s, and each party sends or receives  $(C + n^4) \log n$  bits which over 1Gbps results in a delay of  $\approx 14$  s. If we use [15,23] to implement the broadcast, then the round complexity becomes  $\mathcal{O}(n^2 + D)$  which is  $\approx 8000$  s. The improvement in the round complexity is significant in this scenario. Of course, these are only coarse estimations that do not even take into account the hidden constants in the  $\mathcal{O}$  notation.

## Main Technical Result

Our main result is obtained via several advances in building blocks for perfectly secure optimally resilient MPC. In our view, the most important and technically involved contribution is a new primitive called *Detectable Secret Sharing*. This is a secret sharing with the following properties: (1) Secrecy: The corrupted parties cannot learn anything about the secrets after the sharing phase for an honest dealer; (2) Binding: After the sharing phase (even if the dealer is corrupted), the secret is well defined by the shares of the honest parties; (3) Reconstruction or detection: Reconstruction ends up in the well-defined secret, or it might fail (even if the dealer is honest). However, in the case of failure, there is a

(private) detection of  $\mathcal{O}(n)$  corrupted parties. Moreover, successful sharing and reconstruction are guaranteed if the dealer has already detected more than  $t/2$  corrupted parties before the respective phases.

We show that despite the possible failure of the reconstruction, Detectable Secret Sharing suffices for obtaining our end result for MPC. Most importantly, we obtain a highly efficient construction for this primitive:

**Theorem 1.4** (informal). *There exists a detectable secret sharing protocol that allows sharing  $p$  secrets (of  $\log n$  bits each) with malicious security and optimal resilience with  $\mathcal{O}(n^4 \log n + p \log n)$  communication complexity and expected constant number of rounds.*

For  $p \geq n^4$ , this is  $\mathcal{O}(1)$  field elements per secret (which is also a field element)! This matches packed semi-honest secret sharing as in [28]. The theorem holds for a single dealer; for  $n$  dealers, each sharing  $p$  secrets in parallel, we get  $\mathcal{O}(1)$  field elements per secret starting from  $p \geq n^3$ .

Stated differently, we show a detectable secret sharing protocol that can pack  $\mathcal{O}(n^2)$  secrets (of size  $\log n$  each) at the cost of  $\mathcal{O}(n^2 \log n)$  communication complexity for private channels and each party broadcasts at most  $\mathcal{O}(n \log n)$  bits, with a strictly constant number of rounds. There are at least two striking features of our new detectable secret sharing: *packing*, and *batching*. First, to the best of our knowledge, this is the first protocol in the malicious setting that can pack  $\mathcal{O}(n^2)$  secrets at the cost of  $\mathcal{O}(n^2)$  communication complexity – so an amortized cost of  $\mathcal{O}(1)$  per secret over point-to-point channels. Second, our scheme allows batching –  $m$  independent instances with the same dealer require  $\mathcal{O}(mn^2 \log n)$  over point-to-point channels but just  $\mathcal{O}(n \log n)$  broadcast per party in all  $m$  instances combined. To the best of our knowledge, this is the first protocol that requires a fixed broadcast cost independent of the batching parameter  $m$ . By setting  $m = p/n^2$  and combining with the recent broadcast implementation of Abraham, Asharov, Patil, and Patra [1], we obtain Theorem 1.4 in the point-to-point channel model and no broadcast.

Note that this primitive is formally incomparable with weak-secret sharing [36] (where reconstruction needs the help of the dealer but is guaranteed to succeed when the dealer is honest). On the one hand, our notion seems weaker as there is no guaranteed validity (no guaranteed reconstruction in case of an honest dealer). On the other hand, it is not strictly weaker since our notion ensures mass detection in case of a reconstruction failure. For comparison, the best known weak-secret sharing [2] requires  $\mathcal{O}(n^4 \log n)$  for sharing  $\mathcal{O}(n)$  secrets (i.e.,  $\mathcal{O}(n^3)$  per secret).

**Verifiable Secret Sharing.** We also derive (and use) a “strong” secret sharing (i.e., honest parties always succeed to reconstruct), i.e., in the standard verifiable secret sharing [20] setting:

**Theorem 1.5** (informal). *There exists a protocol that allows to secret share  $p$  secrets (of  $\log n$  bits each) with malicious security and optimal resilience with  $\mathcal{O}(n^4 \log n + p \cdot n \log n)$  communication complexity and expected constant number of rounds.*

For  $p \geq n^3$ , this is an overhead of  $\mathcal{O}(n)$  per secret. Previously, the best known [1] in this setting packs  $\mathcal{O}(n)$  secrets with  $\mathcal{O}(n^4 \log n)$  communication complexity (an overhead of  $\mathcal{O}(n^3)$  per secret). This is an improvement of  $\mathcal{O}(n^2)$  over the state-of-the-art. In comparison, the starting point is the VSS of BGW and Feldman [13, 26] requires  $\mathcal{O}(n^2 \log n)$  point-to-point and  $\mathcal{O}(n^2 \log n)$  broadcast, for sharing just a single secret. This results in  $\mathcal{O}(n^4 \log n)$  communication complexity over point-to-point channels and no broadcast, for sharing just a single secret (an overhead of  $\mathcal{O}(n^4)$ ).

**Detection.** The line of work of [12, 30, 32] in perfectly-secure MPC is based on the *player elimination framework* (introduced by Hirt, Maurer and Przydatek [32]). The protocol identifies a set of parties in which it is guaranteed that one of the players among the set is corrupted, excludes the entire set, and restarts some part of the protocol. The important aspect here is that all parties agree on the set, and that honest parties are also “sacrificed” along the way. In each iteration, the number of parties being excluded is constant. This is a slow process that leads to the  $\mathcal{O}(n)$  rounds overhead.

Instead of globally eliminating a set of parties, our approach is to have each party maintain a local set of conflicted parties, with no global agreement among parties on who is malicious. Each party can decide which parties to mark as conflicted while it shares its own secret(s). When an honest party marks enough corrupt parties as conflicted, its sharing will always be successful. Moreover, whenever there is a failure in sharing or reconstruction, then there is a mass detection –  $\mathcal{O}(n)$  corruptions are identified, either publicly or privately.

To elaborate further, our MPC protocol uses three kinds of detections: (a) *global detection* – wherein a set of parties is excluded from the computation. Unlike [12, 30, 32], in our case, honest parties are never discarded; (b) *public individual detection* – wherein each party has its own conflict set that is publicly known to all. While a similar mechanism, referred to as ‘dispute control’ has been used in [11, 14, 31], these works achieve *statistical* security in the honest majority setting with  $\mathcal{O}(n)$  rounds overhead similar to the player-elimination framework; (c) *private (local) detection* – wherein each party has its private conflict set that it excludes from its local computation. Specifically, an honest party may locally identify a set conflicts (with corrupted parties) without a mechanism to prove that it has done so honestly. In our protocol, it can identify  $\mathcal{O}(n)$  such conflicts simultaneously in case private reconstruction towards it fails. This allows an honest party to locally discard the communication from  $\mathcal{O}(n)$  corrupt parties, eventually ensuring a successful reconstruction.

## 1.1 Related Work

**Broadcast.** Our communication complexity takes into account the cost of broadcast. In the setting of perfect security, there are two families of protocols for implementing the broadcast: once again – efficient and slow, or fast but less efficient. The former [15, 23] takes  $\mathcal{O}(n)$  rounds and  $\mathcal{O}(n^2 + pn)$  for broadcasting a message of  $p$  bits. The latter [1] (built upon Feldman and Micali [25],

and Katz and Koo [33]) takes  $\mathcal{O}(1)$  expected number of rounds and  $\mathcal{O}(n^4 + pn)$  communication complexity for broadcasting a message of size  $p$  bits, i.e., this is optimal for  $p > n^3 \log n$ . Note that when broadcasting a message of size  $p$ , then since each party is supposed to receive  $p$  bits, the minimal possible communication complexity is  $pn$ . Moreover,  $n$  parties broadcasting messages of size  $p$  bits each takes  $\mathcal{O}(n^4 + pn^2)$ , i.e., optimal for  $p > n^2 \log n$ . We also remark that containing strict  $\mathcal{O}(1)$  number of rounds is impossible [27].

**Shunning.** Our notion of detectable secret sharing can be viewed as a synchronous analog of the notion of shunning, in which parties either succeed in their asynchronous verifiable secret sharing or some detection event happens. In the context of asynchronous verifiable secret sharing, shunning was first suggested by Abraham, Dolev, and Halpern [3] and later improved and extended to shunning  $\mathcal{O}(n)$  parties by Bangalore, Choudhury, and Patra [8, 9]. However, unlike our detectable secret sharing, none of these works attain  $\mathcal{O}(1)$  amortized communication cost per secret.

## 2 Technical Overview

In this section, we provide a technical overview of our work. We start in Sect. 2.1 with an overview of our main technical result – our detectable and verifiable secret sharing schemes. In Sect. 2.2 we overview our MPC result. Most of the building blocks are based on previous works, and we highlight in the overview the steps where we made significant improvements. In Sect. 2.3 we overview another step in the protocol, triplet secret sharing.

### 2.1 Detectable and Verifiable Secret Sharing

We start this overview with the most basic verifiable secret sharing protocol – the one by BGW [13]. See also [4, 6] for further details. To share a secret  $s$ , the dealer chooses a bivariate polynomial  $S(x, y) = \sum_{k=0}^t \sum_{\ell=0}^t s_{k,\ell} \cdot x^k y^\ell$  of degree  $t$  in both  $x$  and  $y$  under the constraint that  $S(0, 0) = s_{0,0} = s$ . The share of each party  $P_i$  is the pair of degree- $t$  univariate polynomials  $S(x, i), S(i, y)$ . The goal of the verification step is to verify that the shares of all honest parties indeed lie on a unique bivariate polynomial  $S(x, y)$ . Let us briefly recall the sharing phase:

1. **Sharing:** The dealer sends the share  $(f_i(x), g_i(y)) = (S(x, i), S(i, y))$  to each party  $P_i$ .
2. **Pairwise checks:**  $P_i$  sends to each  $P_j$  the two points  $(f_i(j), g_i(j)) = (S(j, i), S(i, j)) = (g_j(i), f_j(i))$ . If  $P_i$  did not receive from  $P_j$  the points it expects to see (i.e., that agree with  $f_i(x), g_i(y)$ ), then it publicly broadcasts a complaint  $\text{complaint}(i, j, f_i(j), g_i(j))$ .



3. **Publicly resolving the complaints:** The dealer checks all complaints; if some party  $P_i$  publicly complains with values that are different than what the dealer has sent it, then the dealer makes the share of  $P_i$  public – i.e., it broadcasts  $\text{reveal}(i, S(x, i), S(i, y))$ .
4. If a party  $P_j$  sees that (1) all polynomials that the dealer made public agree with its private shares; (2) its share was not made public; (3) if two parties  $P_k$  and  $P_\ell$  both complaint on each other, then the dealer must open one of them. If all those conditions hold, then  $P_j$  is happy with its share, and votes to accept the dealer. If the shares of  $P_j$  were made public, then it re-assigns  $f_j(x), g_j(y)$  to the publicly revealed ones.
5. If  $2t + 1$  parties votes to accept the shares, then each party output its share. Otherwise, the dealer is discarded.

Observe that if the dealer is honest, then during the verification phase the corrupted parties do not learn anything new. Specifically, a party always broadcasts a complaint with the values that it received from the dealer, and the dealer makes a share public only if the public complaint does not contain the values that it has sent that party. Therefore, an honest dealer never makes the shares of an honest party public. Moreover, all honest parties are happy, and accept the shares.

If the dealer is corrupted, then  $2t + 1$  parties that voted to accept the dealer implies that we have a set  $J \subseteq [n]$  of at least  $t + 1$  honest parties that are happy with their shares and that their shares were never made public. The shares of those  $t + 1$  honest parties fully determine a bivariate polynomial of degree- $t$  in both variables. If some honest party  $P_j$  initially held a share that does not agree with this bivariate polynomial, i.e., does not agree with some  $P_k$  for  $k \in J$ , then it must be that  $P_j$  and  $P_k$  both publicly complained, and that the share of  $P_j$  was made public with some new share that agrees with  $S$  (if it does not agree with  $S$ , then at least one party in  $J$  would have not voted to accept). Therefore, at the end, all honest parties hold shares of a well-defined bivariate polynomial.

To reconstruct the bivariate polynomial, each party sends to each other party its pair of polynomials. Since the underlying polynomial is of degree- $t$ , the adversary controls at most  $t$  parties, we must have  $n - t \geq 2t + 1$  correct points and at most  $t$  errors. The Reed-Solomon decoding procedure guarantees that the  $t$  errors can be identified and corrected.

**Our Improvements.** The above scheme for verifiable secret sharing requires  $\mathcal{O}(n^2 \log n)$  communication over the point-to-point channels, and also the broadcast of  $\mathcal{O}(n^2 \log n)$  bits. This results in total communication complexity of  $\mathcal{O}(n^4 \log n)$  over point-to-point for sharing a single secret. The work of [1] has the same complexity for sharing  $\mathcal{O}(n)$  secrets.

For the same communication complexity, we show how to do detectable secret sharing for  $\mathcal{O}(n^4)$  secrets or to do (standard) verifiable secret sharing for  $\mathcal{O}(n^3)$  secrets. Looking ahead, we improve the basic scheme in the following aspects, each giving a factor of  $\mathcal{O}(n^2)$  improvement for our detectable secret sharing:

**(1) Packing:** The bivariate polynomial  $S(x, y)$  in the basic construction contains only a single secret, located at  $S(0, 0)$ . This is the best possible when sharing a bivariate polynomial of degree- $t$  in both  $x$  and  $y$ : The  $t$  shares of the

corrupted parties, together with the secret, fully determine the bivariate polynomial. In our detectable secret sharing scheme, the dealer shares a bivariate polynomial of degrees greater than  $t$  in *both*  $x$  and  $y$ . This allows planting  $\mathcal{O}(n^2)$  secrets. The verification that all parties hold shares on the same bivariate polynomial is much more challenging because the degrees of all the univariate polynomials are greater than  $t$ . Nevertheless, we obtain binding with asymptotically the same cost as the basic scheme, therefore we already obtain an improvement of  $\mathcal{O}(n^2)$  over the basic scheme.

Moreover, once the degree in both dimensions is greater than  $t$ , then reconstruction might fail because the underlying codeword is of degree greater than  $t$ , and the parties cannot necessarily correct the errors if the adversary does not provide correct shares. Nevertheless, Reed-Solomon decoding guarantees that the honest parties can (efficiently) *identify* whether there is a unique decoding or not. We use this property to also *detect sufficiently many* corrupted parties. This suffices for constructing a detectable secret sharing scheme.

For (standard) verifiable secret sharing, we must make the degree in at least one of the dimensions to be at most  $t$ , to allow to always succeed in correcting errors. This allows us to pack “only”  $\mathcal{O}(n)$  secrets and not  $\mathcal{O}(n^2)$ .

**(2) Batching:** The verification step of [13] requires broadcasting  $\mathcal{O}(n^2)$  field elements by the dealer, and  $\mathcal{O}(n)$  field elements by each party. Hence  $m$  independent instances (with the same dealer) require broadcasting of  $\mathcal{O}(mn^2)$  field elements. First, we *balance* the protocol such that each party broadcasts at most  $\mathcal{O}(n)$  field elements, including the dealer. Second, by designing a sharing protocol that is tailored for achieving cheap batching, *the broadcast cost for  $m$  independent instances remains the same as a single instance, i.e., it requires each party to broadcast  $\mathcal{O}(n \log n)$  bits in all  $m$  executions combined.* By setting  $m = \mathcal{O}(n^2)$  and implementing the broadcast over point-to-point, we get a detectable secret sharing of  $\mathcal{O}(n^4)$  secrets (each is a field element of size  $\mathcal{O}(\log n)$ ) at the cost of  $\mathcal{O}(n^4 \log n)$  communication over the point-to-point channels. This is the second  $\mathcal{O}(n^2)$  improvement over the basic scheme.

**Our Batched and Packed Detectable Secret Sharing Protocol.** For our discussion, assume that the dealer first chooses a polynomial  $S(x, y)$  of degree  $t + t/4$  in  $x$  and degree  $t + t/4$  in  $y$ . We will use different parameters in the actual construction later,<sup>1</sup> but we choose  $t + t/4$  for simplicity of exposition in this overview. Like the basic scheme, the view of the adversary consists of the pair of the univariate polynomials  $S(x, i), S(i, x)$ , for every  $i \in I$ , where  $I \subseteq [n]$  is the set of indices of the corrupted parties (of cardinality at most  $t$ ). This means that the adversary receives at most  $2t(t + t/4 + 1) - t^2$  values, and therefore the dealer can still plant  $(t/4 + 1)^2 \in \mathcal{O}(n^2)$  secrets in  $S(x, y)$ , which is fully determined by  $(t + t/4 + 1)^2$  values. Concretely, it can plant for every  $a \in \{0, \dots, t/4\}$  and  $b \in \{0, \dots, t/4\}$  a secret at location  $S(-a, -b)$ .

Looking ahead, to allow *batching*, the dealer will choose  $m$  different bivariate polynomials  $S_1(x, y), \dots, S_m(x, y)$ , and all the parties will verify all the  $m$

<sup>1</sup> Our actual parameters are further optimized to pack more secrets.

instances simultaneously. To accept the shares, all instances must end up successfully. We follow the following two design principles:

1. *Broadcast is expensive; Each broadcast must be utilized in all  $m$  instances, not just in one instance.*
2. *Detection: Whenever a party is detected as an obstacle for achieving agreement (a foe), we should make it a “friend”, or more precisely, we neutralize its capacity to obstruct further, and utilize it to achieve agreement on a later stage.*

We focus on sharing of one instance for now, while keeping these design principles in mind. Along the way, we also discuss how to keep the broadcasts of the dealer low for all  $m$  instances simultaneously, and we will show how to reduce the broadcasts of other parties later on. We follow a similar structure to that of the basic scheme:

1. **Sharing:** The dealer sends  $f_i(x), g_i(y)$  to each party  $P_i$ .
2. **Pairwise checks:** Each pair of parties exchange sub-shares. In case of a mismatch, a party broadcasts a complaint  $\text{complaint}(i, j, f_i(j), g_i(j))$ .

The dealer now has to resolve the complaints. In the basic protocol, when the dealer identifies party  $P_i$  as corrupted, the dealer simply broadcasts the “correct”  $(S(x, i), S(i, y))$  so that everyone can verify that the shares are consistent. However, this leads to  $\mathcal{O}(n^2)$  values being broadcasted, and  $\mathcal{O}(mn^2)$  values in the batched case. Instead, in our protocol, the dealer just marks  $P_i$  as corrupted and adds it to a set  $\text{CONFLICTS} \subset [n]$  which is initially empty. It broadcasts the set  $\text{CONFLICTS}$ . This set should be considered as “parties that had false complaints” from an honest dealer’s perspective. There are three cases to consider:

1. The dealer is discarded: This might happen, e.g., if two parties complained on each other and none of them is in  $\text{CONFLICTS}$ . In this case, it is clear that the dealer is corrupted, and all parties can just discard it.
2. If the dealer is not discarded and  $|\text{CONFLICTS}| > t/4$ , then we have **large conflict**. The dealer identified a large set of conflicts (note that if the dealer is honest, then  $\text{CONFLICTS}$  contains only corrupted parties). Instead of publicly announcing the polynomials  $f_i(x), g_i(y)$  of the identified corrupted parties, the dealer simply restarts the protocol. In the new iteration, the shares of parties in  $\text{CONFLICTS}$  are publicly set to 0. That is, it chooses a new random bivariate polynomial  $S(x, y)$  that hides the same secrets as before, this time under the additional constraints that  $S(x, i) = S(i, y) = 0$  for every  $i \in \text{CONFLICTS}$ .

The dealer does not broadcast the shares of parties in  $\text{CONFLICTS}$ ; all the parties know that they are 0s. When each party receives its new pair of shares  $f_j(x), g_j(y)$ , it also verifies that  $f_j(i) = g_j(i) = 0$ , and if not, it raises a complaint. Parties in  $\text{CONFLICTS}$  cannot raise any complaints. Furthermore, observe that the outcome of “large conflict” might occur only  $\mathcal{O}(1)$  times; if the dealer tries to exclude more than  $t$  parties total, then the dealer is publicly discarded.

When batching over  $m$  instances, we choose the shares of the set **CONFLICTS** to be 0 in all instances. Thus, the dealer uses a broadcast of  $\mathcal{O}(n \log n)$  bits, i.e., the set **CONFLICTS**, and by restarting the protocol it made the shares of parties in **CONFLICTS** public in all  $m$  executions. Thus, we get the same effect as broadcasting  $m|\mathbf{CONFLICTS}|$  pairs of polynomials (i.e., broadcasting  $\mathcal{O}(m \cdot n^2 \log n)$  bits). This follows exactly our first design principle.

3. If  $|\mathbf{CONFLICTS}| \leq t/4$  then the dealer proceeds with the protocol. It has to reconstruct the  $f$  and  $g$  polynomials of all parties in **CONFLICTS**.

Before we proceed, let's highlight what guarantees we have so far: when the dealer is honest, then all the parties in **CONFLICTS** are corrupted. Moreover, if in some iteration there were more than  $t/4$  identified conflicts by the dealer, those corrupted parties are eliminated, and they have shares that all parties know (i.e., 0) and are consistent with the shares of the honest parties. This turns a “foe” into a “friend”, as our second design principle.

When the dealer is corrupted, then all parties that are not in **CONFLICTS** have shares that define a unique bivariate polynomial, and we have binding. Specifically, if the shares of two honest parties do not agree with each other, then they both complain on each other, and the dealer must include one of them in **CONFLICTS**. Therefore, all honest parties that are not in **CONFLICTS** (assuming that the dealer was not publicly discarded) hold shares that are consistent with each other. Moreover, there is one more important property: Honest parties that were excluded in previous iterations (and now their shares are 0) also hold shares that are consistent with the honest parties that are not in **CONFLICTS**. In particular, if we indeed proceed, then there are at most  $t/4$  honest parties who do not hold shares on the polynomial. This means that there are  $2t + 1 - t/4$  honest parties that have shares on the bivariate polynomial – not only do we have binding, but we also have some redundancy! This redundancy will be crucial for our next step as we show below.

However, there might still be up to  $t/4$  honest parties (in **CONFLICTS**) that do not have shares on the correct polynomial. The rest of the protocol is devoted to reconstructing their shares. We call this phase reconstruction of the shares of honest parties in **CONFLICTS**. However, before proceeding to the reconstruction, we first describe how to batch over  $m$  instances.

**Batching Complaints.** Consider sharing  $m$  instances simultaneously with the same dealer. In the above description, we already described how the dealer's broadcast is just the set **CONFLICTS**, which require  $\mathcal{O}(n \log n)$  bits, independent of  $m$ . However, the broadcast of other parties depends on  $m$ . Specifically:

1. A party  $P_i$  broadcasts  $\mathbf{complaint}(i, j, f_i(j), g_i(j))$  when it receives a wrong share from some party  $P_j$ .
2. A party  $P_i$  broadcast  $\mathbf{complaint}$  if the share it received do not agree with the parties that are publicly 0. Recall that in that case, the dealer must include  $P_i$  in **CONFLICTS**.

It suffices to complain in only one of the instances, say the one with the lexicographically smallest index. This follows our first design principle. If two parties

$P_i$  and  $P_j$  do not agree in  $\ell < m$  of the instances, they will both file a joint complaint with the same minimal index. Thus, we have a joint complaint, and in order to not be discarded, the dealer must include either  $i$  or  $j$  in CONFLICTS. Thus, we still have the guarantee that if two honest parties are not in CONFLICTS then their shares must be consistent, now in *all*  $m$  executions.

Likewise, if some party  $P_i$  receives from the dealer private shares where on points of some parties that were excluded it does not receive 0s, it essentially requires to be part of CONFLICTS. Thus, there is no need to make  $m$  requests, it suffices to make just one such request.

**Reconstruction of the Shares of Honest Parties in CONFLICTS.** Going back to the last step of the sharing process, each party  $P_j$  in CONFLICTS wishes to reconstruct its pair of polynomials  $(f_j(x), g_j(y))$ . Towards that end, each party  $P_k$  that is not in CONFLICTS sends to  $P_j$ , privately, the values  $(f_j(k), g_j(k))$ .  $P_j$  therefore is guaranteed to receive  $2t + 1 - t/4$  correct points. However, the polynomials are of degree  $t + t/4$ , and we need  $2t + t/4 + 1$  “correct values” to eliminate  $t$  errors. This means that if the adversary introduces more than  $t/2$  incorrect values,  $P_j$  does not have unique decoding. In this case,  $P_j$  broadcasts a complaint  $\text{complaint}(j)$ , insisting that its shares be publicly reconstructed. As we will see, when batching over  $m$  executions, it is enough to make one public complaint in one execution, say the lexicographically smallest one, let’s denote it as  $\beta \in [m]$ . Resolving this instance will help to resolve all other  $m$  instances.

Upon receiving  $\text{complaint}(j, \beta)$ , each party  $P_k$  broadcasts  $\text{reveal}(k, j, f_k(j), g_k(j))$  for the  $\beta$ th instance. Thus, we will have at least  $2t + 1 - t/4$  correct values that are public. Moreover, corrupted parties might now reveal values that are different than what they have previously sent privately, and we might already have unique decoding. In any case, with each value that was broadcasted and is wrong, the dealer adds the identity of the party that broadcasted the wrong value into a set **Bad**. It then broadcasts the set **Bad**, and all parties can check that when excluding parties in **Bad** then all other values define a unique polynomial, and all public points (excluding **Bad**) lie on this polynomial. Otherwise, the dealer is publicly discarded. Note that it is enough to broadcast one set **Bad** for all party  $j \in \text{CONFLICTS}$  and for all  $m$  instances. If  $|\text{Bad}| > t/2$ , we restart the protocol, again giving shares 0 to parties in **Bad** (as long as the total number of parties that the dealer excluded does not exceed  $t$ ).

At this point, if we did not restart and the dealer was not discarded, then it must be that  $P_j$  can reconstruct its polynomials  $f_j(x), g_j(y)$  in *all*  $m$  instances. First, in the  $\beta$ th instance (that was publicly resolved), we know that we have  $2t + 1 - t/4$  public points that are “correct” and that the dealer could have excluded at most  $t/2$  parties. Therefore, there are more than  $t + t/4 + 1$  correct points even if the dealer excludes up to  $t/2$  honest parties (recall that it cannot exclude more than  $t/2$ ). Those correct points uniquely determine a polynomial of degree  $t + t/4$ , and therefore, since all points after excluding parties in **Bad** lie on one unique polynomial, it must be that this polynomial is the correct one.

Using the information learned in the resolved instance party  $P_j$  can uniquely decode all other  $m$  instances. Specifically, there is no unique decoding in a par-

ticular instance only if  $P_j$  received more than  $t/2$  wrong private shares. When going publicly, some parties might announce different values than what they first told  $P_j$  privately.  $P_j$  can compare between the polynomial reconstructed in the  $\beta$ th instance to the initial values it received privately from the parties, and identify all parties that sent it wrong shares. Denote this set as  $\text{localBad}_j$ . It must hold that this set contains more than  $t/2$  corrupted parties. Now, in each one of the other instances, ignore all parties in  $\text{localBad}_j$ . This implies that the remaining values are of distance at most  $t/2$  from a correct word, i.e., they contain at most  $t/2$  errors. Moreover, it is guaranteed that honest parties are not eliminated, and we still have at least  $2t + 1 - t/4$  correct points. Therefore,  $P_j$  guarantees to have unique decoding in all  $m$  instances.

**Detectable and Robust Reconstruction.** So far, we described the sharing procedure. While we do not use the reconstruction of detectable secret sharing directly (we will use private reconstruction, and parties never reconstruct all secrets), we briefly describe it for completeness. To reconstruct polynomials  $S_1(x, y), \dots, S_m(x, y)$  that were shared with the same dealer, we follow a similar step as reconstruction towards parties in CONFLICTS, but with reconstructing all polynomials: Each party sends (privately) the  $f$ -shares, the parties try to privately reconstruct  $g_i$ -polynomials for all  $i \in [n]$ , and interpolate the bivariate polynomials from the  $g_i$ -polynomials. If some party does not succeed in uniquely reconstructing some  $g_i$ -polynomial, then it asks to go public. For each party  $P_j$ , it is enough to publicly reconstruct one  $g_i$ -polynomial that it did not succeed to reconstruct privately, and from that,  $P_j$  can reconstruct all other shares (by ignoring the new privately detected parties).

However, as before, the adversary can cause the reconstruction to fail. When it does so, the dealer is guaranteed to detect more than  $t/2$  corruptions. Moreover, if the dealer already detected at least  $t/2$  corruptions during the sharing phase, then those parties cannot fail the reconstruction, and reconstruction is guaranteed. Note that the cost of the reconstruction is  $\mathcal{O}(mn^2 \log n)$  over point-to-point channels, plus each party has to broadcast at most  $\mathcal{O}(n \log n)$  bits, again, independent of  $m$ .

**Reconstruction for VSS.** Recall that for VSS, we set the degree of  $y$  in each bivariate polynomial to  $t$ . This implies that all parties can reconstruct all  $g$ -polynomials using Reed-Solomon error correction and we never have to resolve complaints publicly. Moreover, the adversary can never cause any failure. The cost is therefore  $\mathcal{O}(mn^2 \log n)$  over point-to-point channels, and VSS robust reconstruction is always guaranteed.

We refer the reader to Sect. 4 for our packed secret sharing scheme for a single polynomial, and to Sect. 5 for the batched version.

## 2.2 Our MPC Protocol

Our MPC protocol follows the following structure: an offline phase in which the parties generate Beaver triplets [10], and an online phase in which the parties compute the circuit while consuming those triples.

**Beaver triplets generation.** Our goal is to distribute shares of random secret values  $a, b$  and  $c$ , such that  $c = ab$ . If the circuit contains  $C$  multiplication gates, then we need  $C$  such triplets. Towards that end, we follow the same steps as in [22], and generate such triplets in two stages:

1. **Triplets with a Dealer:** Each party generates shares of  $a_i, b_i, c_i$  such that  $c_i = a_i \cdot b_i$ . We generate all the triplets in parallel using expected  $\mathcal{O}(1)$  rounds. We will elaborate on this step below in Sect. 2.3. Our main contribution is in improving this step. In our protocol, each party acts as a dealer to generate  $mn$  triplets. This step requires an overall cost of  $\mathcal{O}(n^4 \log n + mn^3 \log n)$  point-to-point communication for all the parties together. Later, these  $mn^2$  triplets will be used for generating  $\mathcal{O}(mn^2)$  triplets overall. Looking ahead, we will use  $m = C/n^2$  and this step costs  $\mathcal{O}(n^4 \log n + Cn \log n)$ . Previously, the best known [22] used  $\mathcal{O}(n^3 \log n)$  point-to-point and  $\mathcal{O}(n^3 \log n)$  broadcast for generating just a single triplet for one dealer. That is, for  $\mathcal{O}(mn^2)$  triplets this is  $\mathcal{O}(mn^5 \log n)$  broadcast which costs at least  $\Omega(mn^6 \log n)$  over point-to-point. We therefore improve in a factor of  $\mathcal{O}(n^3)$ .
2. **Triplets with No Dealer:** Using triplet extraction of [22], we can extract from a total of  $C$  triplets with a dealer,  $\mathcal{O}(C)$  triplets where no party knows the underlying values. That is, if  $n$  parties generate  $C/n$  triplets each, then we have a total of  $C$  triplets and we can extract from it  $\mathcal{O}(C)$  triplets. This step costs  $\mathcal{O}(n^2 \log n + Cn \log n)$ .

Putting it all together, for generating  $C$  triplets we pay a total of  $\mathcal{O}(n^4 \log n + Cn \log n)$  and constant expected number of rounds.

The MPC protocol then follows the standard structure where each party shares its input, and the parties evaluate the circuit gate-by-gate, or more exactly, layer-by-layer. In each multiplication gate, the parties have to consume one multiplication triple. Using the method of [22], if the  $i$ th layer of the circuit contains  $C_i$  multiplications (for  $i \in [D]$ , where  $D$  is the depth of the circuit), the evaluation costs  $\mathcal{O}(n^2 \log n + C_i \cdot n \log n)$ . Summing over all layers, this is  $\sum_{i \in [D]} (n^2 + nC_i) \log n = (Dn^2 + Cn) \log n$ . Together with the generation of the triplets, we get the claimed  $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$  cost as in Theorem 1.3. We refer the readers to the full version for further details on our MPC protocol.

### 2.3 Multiplication Triplets with a Dealer

As mentioned, a building block which we improve in a factor of  $\mathcal{O}(n^3)$  over the state-of-the-art is multiplication triplets with a dealer. The goal is that given a dealer, to distribute shares of secret values  $\mathbf{a}, \mathbf{b}, \mathbf{c}$  such that for every  $i$  it holds that  $c_i = a_i b_i$ . Towards this end, the dealer plants  $\mathbf{a}$  into some bivariate polynomial  $A(x, y)$  using our verifiable secret sharing scheme. It plants  $\mathbf{b}$  into  $B(x, y)$  and  $\mathbf{c}$  into  $C(x, y)$  in a similar manner. Note that we use verifiable secret sharing here, since we want to output the triplets shared via degree- $t$  polynomials (which is utilized by our MPC protocol). So we can plant only  $\mathcal{O}(n)$  values in each one of them. Then, the dealer has to prove, using a distributed zero-knowledge protocol, that indeed  $c_i = a_i b_i$  for every  $i$ . The zero-knowledge proof

uses sharing and computations on the coefficients of the polynomials used for sharing  $\mathbf{a}, \mathbf{b}, \mathbf{c}$ , i.e., if we shared  $\mathcal{O}(M)$  triplets, then the zero-knowledge involves sharing of  $\mathcal{O}(Mn)$  values. However, since the dealer is involved in the sharing and the reconstruction of those values, we do not need full-fledged secret sharing scheme, and we can use the lighter detectable secret sharing. This scheme enables us to share  $\mathcal{O}(Mn)$  values at the same cost of “strong” verifiable secret sharing of  $\mathcal{O}(M)$  values.

In a more detail, after verifiable sharing  $A, B$  and  $C$  each of degree  $t + t/4$  in  $x$  and  $t$  in  $y$ , the dealer needs to prove that for every  $a \in \{0, \dots, t/4\}$  it holds that  $C(-a, 0) = A(-a, 0) \cdot B(-a, 0)$ . Towards that end, for every  $a \in \{0, \dots, t/4\}$  it considers the polynomial

$$E_{-a}(y) = A(-a, y) \cdot B(-a, y) - C(-a, y) = e_{-a,0} + e_{-a,1}y + \dots + e_{-a,2t}y^{2t}$$

and its goal is to show that the degree- $2t$  polynomial  $E_{-a}(y)$  evaluates to 0 on each  $y \in \{0, \dots, -t/4\}$ . The dealer secret-shares all the coefficients  $(e_{-a,k})$  for  $a \in \{0, \dots, t/4\}$  and  $k \in \{0, \dots, 2t\}$  using our detectable secret sharing scheme, by packing them into several bivariate polynomials  $E(x, y)$  with degree  $t + t/4$  in both  $x$  and  $y$ . Note that there are  $\mathcal{O}(n^2)$  coefficients to share, and each polynomial  $E(x, y)$  can pack  $(t/4 + 1)^2$  secrets.<sup>2</sup> Thus, we actually share a constant number (precisely 8) of polynomials to share all the coefficients.

Using linear combinations over the shares, the reconstruction protocol privately reconstructs towards  $P_j$  (for each  $j \in [n]$ ) the evaluation of  $E_{-a}(y)$  on  $j$ , i.e.,  $E_{-a}(j)$ , for each  $a \in \{0, \dots, t/4\}$ . This is performed in a similar manner to the reconstruction of shares of honest parties in CONFLICTS in our detectable secret sharing protocol. Each  $P_j$  can then verify that  $E_{-a}(j) = A(-a, j) \cdot B(-a, j) - C(-a, j)$ , and if not, it can raise a public complaint. Parties can then open the shares of  $P_j$  on  $A, B, C$  publicly, and also the value  $E_{-a}(j)$ . If indeed  $E_{-a}(j) \neq A(-a, j) \cdot B(-a, j) - C(-a, j)$ , then the dealer is discarded.

Moreover, again using linear evaluations over the shares and reconstruction, the parties can obtain  $E_{-a}(0)$  for every  $a \in \{0, \dots, t/4\}$  and verify that it equals 0. If indeed  $E_{-a}(j) = A(-a, j) \cdot B(-a, j) - C(-a, j)$  for  $2t + 1$  such  $js$ , then  $E_{-a}(y) = A(-a, y) \cdot B(-a, y) - C(-a, y)$  as those are two polynomials of degree  $2t$  that agree on  $2t + 1$  points. Moreover, if indeed  $E_{-a}(0) = 0$  for every  $a \in \{0, \dots, t/4\}$ , then  $C(-a, 0) = A(-a, 0) \cdot B(-a, 0)$  for every  $a \in \{0, \dots, t/4\}$ , as required.

The above description is a bit oversimplified. Recall that the coefficients of  $E$  are shared using only detectable secret sharing. This means that the private reconstruction towards some  $P_j$  might fail. In that case,  $P_j$  will ask to perform public reconstruction, and the adversary learns  $E_{-a}(j)$  on a point  $j \notin I$ . This is a leakage because the reconstruction was meant to be private and becomes public. The good news is that the outcome of each such public reconstruction is that party  $P_j$  identifies at least  $t/2$  corruptions in  $\text{localBad}_j$ , and all the later reconstructions towards it must succeed.

<sup>2</sup> Again, in the actual construction we will use different dimensions, but we keep using a bivariate polynomial with degree  $t + t/4$  in both  $x$  and  $y$  for simplicity.



As a result, the adversary may learn up to  $n-t$  reconstructions that it was not supposed to learn. Whenever this occurs, we cannot use the entire polynomials that are involved (which pack  $\mathcal{O}(n)$  triplets). If a “pack” of triplets requires a public reconstruction, we discard the whole “pack”. On the positive side, this can happen at most once per party. Moreover, since the multiplication triplets are just random and do not involve secret inputs, we can just sacrifice them. This means that for generating  $m$  “packs” of triplets, we need to start with batching  $\mathcal{O}(m+n)$  “packs” of triplets. This additional overhead does not affect the overall complexity, but it makes the functionalities and the protocol a bit more involved. We refer the reader to Sect. 6 for further details.

**Organization.** The rest of this paper is organized as follows. After some Preliminaries (Sect. 3) we focus on our packed (Sect. 4) and batched (Sect. 5) secret sharing. We then discuss our multiplication triplets with a dealer (Sect. 6), and conclude with the MPC protocol in Sect. 7. Due to lack of space, the proofs and some constructions are deferred to the full version.

### 3 Preliminaries

**Network model and definitions.** We consider a synchronous network model where the parties in  $\mathcal{P} = \{P_1, \dots, P_n\}$  are connected via pairwise private and authenticated channels. Additionally, for some of our protocols we assume the availability of a broadcast channel, which allows a party to send an identical message to all the parties. The distrust in the network is modelled as a *computationally unbounded* active adversary  $\mathcal{A}$  which can maliciously corrupt up to  $t$  out of the  $n$  parties during the protocol execution and make them behave in an arbitrary manner. We prove security in the stand-alone model for a static adversary. We provide the definitions (which are standard) in the full version. Owing to the results of [18], this guarantees adaptive security with inefficient simulation. We derive universal composability [16] using [34].

Our protocols are defined over a finite field  $\mathbb{F}$  where  $|\mathbb{F}| > n + t/2 + 1$ . We denote the elements by  $\{-t/2, -t/2 + 1, \dots, 0, 1, \dots, n\}$ . We use  $\langle v \rangle$  to denote the degree- $t$  Shamir-sharing of a value  $v$  among parties in  $\mathcal{P}$ .

**Bivariate Polynomials and Secret Embedding.** A degree  $(l, m)$ -bivariate polynomial over  $\mathbb{F}$  is of the form  $S(x, y) = \sum_{i=0}^l \sum_{j=0}^m b_{ij} x^i y^j$  where  $b_{ij} \in \mathbb{F}$ . The polynomials  $f_i(x) = S(x, i)$  and  $g_i(y) = S(i, y)$  are called  $i$ th  $f$  and  $g$  univariate polynomials of  $S(x, y)$  respectively. In our protocol, we use  $(t+t/2, t+d)$ -bivariate polynomials where  $d \leq t/4$ , and the  $i$ th  $f$  and  $g$  univariate polynomials are associated with party  $P_i$  for every  $P_i \in \mathcal{P}$ .

We view a list of  $(t/2 + 1)(d + 1)$  secrets SECRETS as a  $(t/2 + 1) \times (d + 1)$  matrix. We then say that the set SECRETS is *embedded* in a bivariate polynomial  $S(x, y)$  of degree  $(t + t/2)$  in  $x$  and  $(t + d)$  in  $y$  if for every  $a \in \{0, \dots, t/2\}$  and  $b \in \{0, \dots, d\}$  it holds that  $S(-a, -b) = \text{SECRETS}(a, b)$ .

**Simultaneous Error Correction and Detection of Reed-Solomon Codes.** We require the following coding-theory related results. Let  $C$  be an

Reed-Solomon (RS) code word of length  $N$ , corresponding to a  $k$ -degree polynomial (containing  $k + 1$  coefficients). Assume that at most  $t$  errors can occur in  $C$ . Let  $\bar{C}$  be the word after introducing error in  $C$  in at most  $t$  positions. Let the distance between  $C$  and  $\bar{C}$  be  $s$  where  $s \leq t$ . Then there exists an *efficient* decoding algorithm that takes  $\bar{C}$  and a pair of parameters  $(e, e')$  as input, such that  $e + e' \leq t$  and  $N - k - 1 \geq 2e + e'$  hold and gives one of the following as output:

1. Correction: output  $C$  if  $s \leq e$ , i.e. the distance between  $C$  and  $\bar{C}$  is at most  $e$ ;
2. Detection: output “more than  $e$  errors” otherwise.

Note that detection does not return the error indices, rather it simply indicates error correction fails due to the presence of more than correctable (i.e.  $e$ ) errors. The above property of RS codes is traditionally referred to as *simultaneous error correction and detection*. In fact the bounds,  $e + e' \leq t$  and  $N - k - 1 \geq 2e + e'$ , are known to be necessary. We cite:

**Theorem 3.1** ([21, 35]). *Let  $C$  be an Reed-Solomon (RS) code word of length  $N$ , corresponding to a  $k$ -degree polynomial (containing  $k + 1$  coefficients). Let  $\bar{C}$  be a word of length  $N$  such that the distance between  $C$  and  $\bar{C}$  is at most  $t$ . Then RS decoding can correct up to  $e$  errors in  $\bar{C}$  to reconstruct  $C$  and detect the presence of up to  $e + e'$  errors in  $\bar{C}$  if and only if  $N - k - 1 \geq 2e + e'$  and  $e + e' \leq t$ .*

A couple of corollaries follows from the above theorem that we will use in our work, see the full version for details.

**Parallel Broadcast.** In our MPC, we use parallel broadcast that relates to the case where  $n$  parties wish to broadcast a message of size  $L$  bits in parallel, as captured in the following functionality.

---

**Functionality 3.2:**  $\mathcal{F}_{BC}^{\text{parallel}}$

---

The functionality is parameterized with a parameter  $L$ .

1. Each  $P_i \in \mathcal{P}$  sends the functionality its message  $M_i \in \{0, 1\}^L$ .
  2. The functionality sends to all parties the message  $\{M_i\}_{i \in [n]}$ .
- 

The work of [1] presents an instantiation with the following security and complexity. Also note that, when some party has smaller message than  $L$  bits, it can pad with default values to make an  $L$  bit message.

**Theorem 3.3** ([1]). *There exists a perfectly-secure parallel broadcast with optimal resilience of  $t < n/3$ , which allows  $n$  parties to broadcast messages of size  $L$  bits each, at the cost of  $\mathcal{O}(n^2L)$  bits communication, plus  $\mathcal{O}(n^4 \log n)$  expected communicating bits. The protocols runs in constant expected number of rounds.*

## 4 Packed Secret Sharing

In this section we present our secret sharing scheme. In the introduction, we mentioned that we have two variants: regular verifiable secret sharing, and a novel detectable secret sharing. The protocol presented in this section fits the two primitives, where the difference is obtained by using different parameters in the bivariate polynomial, as we will see shortly. In this section, we still do not “batch” over multiple polynomials; the dealer share just a single polynomial. In Sect. 5 we provide details on the batched version. The packed secret sharing protocol consists of the following building blocks:

1. The dealer chooses a bivariate polynomial  $S(x, y)$  of degree  $3t/2$  in  $x$  and degree  $t + d$  in  $y$ , where its secret are embedded in  $S$ . We should think of  $d$  as 0 or  $t/4$ . Unlike presented in Sect. 2.1, we have two different parameters for  $x$  and  $y$ . Looking ahead, for verifiable secret sharing, we use  $d = 0$ . For detectable secret sharing, we can use  $d \in [1, t/4]$  (packing  $\mathcal{O}((d+1)n)$  secrets).
2. The dealer tries to share  $S(x, y)$  using a functionality called  $\mathcal{F}_{\text{ShareAttempt}}$  (see Functionality 4.1). At the end of this functionality, the sharing attempt might have the following three outcomes: (a) **discard** – the dealer is discarded; (b) (**detect**, **CONFLICTS**) - a large set of conflicts was detected and the protocol will be restarted; (c) **proceed**, in which case all parties also receive a set **CONFLICTS** (of size at most  $t/2 - d$ ) of parties that still did not receive shares. All honest parties not in **CONFLICTS** hold shares that define unique bivariate polynomial of the appropriate degree. See Sect. 4.1 for further details.
3. The goal is now to let parties in **CONFLICTS** to learn their shares. Since the degrees of the bivariate polynomial is not symmetric, we first reconstruct the  $g$ -share (of degree  $t + d < 3t/2$ ), and then the  $f$ -share (of degree  $3t/2$ ). Reconstruction of  $g$ -polynomial is described in Sect. 4.2. The reconstruction of  $f$ -polynomial is similar, and is discussed in Sect. 4.3.

We first present the different building blocks, and then in Sect. 4.4 we provide the protocol (and functionality) for packed secret sharing, that uses those building blocks.

### 4.1 Sharing Attempt

We start with the description of the functionality.

**Functionality 4.1: Sharing Attempt**–  $\mathcal{F}_{\text{ShareAttempt}}$

The functionality is parameterized with the set of corrupted parties  $I \subset [n]$ .

1. All the honest parties send to  $\mathcal{F}_{\text{ShareAttempt}}$  a set  $\text{ZEROS} \subset [n]$ . For an honest dealer, it holds that  $\text{ZEROS} \subseteq I$ .  $\mathcal{F}_{\text{ShareAttempt}}$  sends the set  $\text{ZEROS}$  to the adversary.
2. The dealer sends a polynomial  $S(x, y)$  to  $\mathcal{F}_{\text{ShareAttempt}}$ . When either the polynomial is not of degree at most  $3t/2$  in  $x$  and at most  $t + d$  in  $y$ , or for some  $i \in \text{ZEROS}$  it holds that  $S(x, i) \neq 0$  or  $S(i, y) \neq 0$ ,  $\mathcal{F}_{\text{ShareAttempt}}$  executes Step 4c to discard the dealer.
3. For every  $i \in I$ ,  $\mathcal{F}_{\text{ShareAttempt}}$  sends  $(S(x, i), S(i, x))$  to the adversary. It receives back a set  $\text{CONFLICTS}$  such that  $\text{CONFLICTS} \cap \text{ZEROS} = \emptyset$ .<sup>3</sup> If the dealer is honest, then  $\text{CONFLICTS} \cup \text{ZEROS} \subseteq I$ . If  $|\text{CONFLICTS} \cup \text{ZEROS}| > t$  for a corrupt dealer, then  $\mathcal{F}_{\text{ShareAttempt}}$  executes Step 4c to discard the dealer.
4. **Output:**
  - (a) Detect: If  $|\text{CONFLICTS}| > t/2 - d$ , then send  $(\text{detect}, \text{CONFLICTS})$  to all parties.
  - (b) Proceed: Otherwise, send  $(\text{proceed}, S(x, i), S(i, y), \text{CONFLICTS})$  for every  $i \notin \text{CONFLICTS}$  and  $(\text{proceed}, \perp, \perp, \text{CONFLICTS})$  to every  $i \in \text{CONFLICTS}$ .
  - (c) Discard: send  $\text{discard}$  to all parties.

**Protocol 4.2: Sharing Attempt**–  $\Pi_{\text{ShareAttempt}}$

**Common Input:** The description of a field  $\mathbb{F}$ , parameter  $d < t$ .

**Input:** All parties input  $\text{ZEROS} \subset [n]$ . The dealer inputs a polynomial  $S(x, y)$  with degree  $3t/2$  in  $x$  and  $t + d$  in  $y$ , such that for every  $i \in \text{ZEROS}$  it holds that  $S(x, i) = 0$  and  $S(i, y) = 0$ .

**The protocol:**

1. **(Dealing shares):** The dealer sends  $(f_i(x), g_i(y)) = (S(x, i), S(i, y))$  to  $P_i$  for  $i \notin \text{ZEROS}$ . Each  $P_i$  for  $i \in \text{ZEROS}$  sets  $(f_i(x), g_i(y)) = (0, 0)$ .
2. **(Pairwise Consistency Checks):**
  - (a) Each  $i \notin \text{ZEROS}$  sends  $(f_i(j), g_i(j))$  to every  $j \notin \text{ZEROS}$ . Let  $(f_{ji}, g_{ji})$  be the values received by  $P_i$  from  $P_j$ .
  - (b) Each  $i \notin \text{ZEROS}$  broadcasts  $\text{complaint}(i, j, f_i(j), g_i(j))$  if (a)  $f_{ji} \neq g_i(j)$  or  $g_{ji} \neq f_i(j)$  for any  $j \notin \text{ZEROS}$ . For  $j \in \text{ZEROS}$ ,  $P_i$  broadcasts  $\text{complaint}(i, j, f_i(j), g_i(j))$  if  $f_i(j) \neq 0$  or  $g_i(j) \neq 0$ .
3. **(Conflict Resolution):**
  - (a) The dealer sets  $\text{CONFLICTS} = \emptyset$ . For each  $\text{complaint}(i, j, u, v)$  such that  $u \neq S(j, i)$  or  $v \neq S(i, j)$ , the dealer adds  $i$  to  $\text{CONFLICTS}$ . The dealer broadcasts  $\text{CONFLICTS}$ .
  - (b) Discard the dealer if any one of the following does not hold: (i)  $|\text{ZEROS} \cap \text{CONFLICTS}| = \emptyset$ ; (ii)  $|\text{CONFLICTS} \cup \text{ZEROS}| \leq t$  (iii) if some  $P_i$  broadcasted  $\text{complaint}(i, j, u_i, v_i)$  and  $P_j$  broadcasted  $\text{complaint}(j, i, u_j, v_j)$  with  $u_i \neq v_j$  or  $v_i \neq u_j$ , then  $\text{CONFLICTS}$  should contain either  $i$  or  $j$  (or both); (iv) if some  $P_i$  broadcasted  $\text{complaint}(i, j, u, v)$  with  $j \in \text{ZEROS}$  and  $u \neq 0$  or  $v \neq 0$ , then  $i \in \text{CONFLICTS}$ .
4. **(Output):** Each  $P_i$  outputs  $\text{discard}$  when the dealer is discarded and  $(\text{detect}, \text{CONFLICTS})$  when  $|\text{CONFLICTS}| > t/2 - d$ . Else, it outputs  $(\text{proceed}, \perp, \perp, \text{CONFLICTS})$  when  $i \in \text{CONFLICTS}$ , and  $(\text{proceed}, f_i(x), g_i(y), \text{CONFLICTS})$  otherwise.

<sup>3</sup> To ease understanding and notion, we sometimes expect to receive from the adversary some sets or inputs that satisfy some conditions. We do not necessarily verify the conditions in the functionality, and this is without loss of generality. For instance, in this step we require that the adversary sends a set  $\text{CONFLICTS}$  such that  $\text{CONFLICTS} \cap \text{ZEROS} = \emptyset$ . Instead, we can enforce that this is the case by resetting:  $\text{CONFLICTS} = \text{CONFLICTS} \setminus \text{ZEROS}$ .

**Lemma 4.3.** *Protocol 4.2,  $\Pi_{\text{ShareAttempt}}$ , perfectly-securely computes Functionality 4.1,  $\mathcal{F}_{\text{ShareAttempt}}$ , in the presence of a malicious adversary, controlling at most  $t < n/3$ .*

## 4.2 Reconstruction of $g$ -polynomials in CONFLICTS

When invoking this functionality, we are guaranteed that the shares of the honest parties define a unique bivariate polynomial, and that the number of parties that are not in CONFLICTS is at least  $(n - t/2) + d$ . The goal of this step is to reconstruct the  $g$ -polynomials for the parties in CONFLICTS, while the possible outcomes are: (i) the dealer is discarded; (ii) the dealer detects additional  $t/2$  parties that it will make ZEROS in the next iteration; (iii) the protocol succeeds and all honest parties hold  $g_j(y)$  as output.

---

### Functionality 4.4: Reconstruction of $g$ -Polynomials – $\mathcal{F}_{\text{rec-g}}$

---

1. **Input:**<sup>4</sup> All honest parties send to the functionality  $\mathcal{F}_{\text{rec-g}}$  the sets  $\text{ZEROS} \subset [n]$  and  $\text{CONFLICTS} \subset [n]$ , each honest  $j \notin \text{CONFLICTS}$  sends  $(f_i(x), g_i(y))$ . Let  $S(x, y)$  be the unique bivariate polynomial of degree at most  $3t/2$  in  $x$  and at most  $t + d$  in  $y$  that satisfies  $f_j(x) = S(x, j)$  and  $g_j(y) = S(j, y)$  for every  $j \notin \text{CONFLICTS}$ . Moreover, it holds that  $n - |\text{CONFLICTS}| \geq 2t + 1 + t/2 + d$ .
  2.  $\mathcal{F}_{\text{rec-g}}$  sends  $(\text{ZEROS}, \text{CONFLICTS}, (S(x, i), S(i, y))_{i \in I})$  to the adversary. If the dealer is corrupted, then  $\mathcal{F}_{\text{rec-g}}$  sends  $S(x, y)$  as well.
  3. It receives back from the adversary a message  $M$ .
  4. **Output:**
    - (a) If  $M = \text{discard}$  and the dealer is corrupted, then  $\mathcal{F}_{\text{rec-g}}$  sends **discard** to all parties.
    - (b) If  $M = (\text{detect}, \text{Bad})$  with  $\text{Bad} \cap (\text{ZEROS} \cup \text{CONFLICTS}) = \emptyset$  and  $|\text{Bad}| > t/2$ , and with  $\text{Bad} \subseteq I$  in the case of an honest dealer, then  $\mathcal{F}_{\text{rec-g}}$  sends  $(\text{detect}, \text{Bad})$  to all parties.
    - (c) If  $M = \text{proceed}$ , then  $\mathcal{F}_{\text{rec-g}}$  sends:
      - for each  $j \in \text{CONFLICTS}$  the output  $(\text{proceed}, \perp, S(j, y))$ , and
      - for each  $j \notin \text{CONFLICTS}$  send  $(\text{proceed}, S(x, j), S(j, y))$ .
- 

<sup>4</sup> If not all honest parties send shares that lie on the same bivariate polynomial, or not all send inputs that satisfy the input assumptions as described, then no security is guaranteed. This can be formalized as follows. If the input assumptions do not hold, then the functionality sends to the adversary all the inputs of all honest parties, and lets the adversary to singlehandedly determine all outputs of all honest parties. This makes the protocol vacuously secure (since anything can be simulated).

---

**Protocol 4.5: Reconstruct  $g$ -Polynomials in CONFLICTS –  $\Pi_{\text{rec-g}}$** 


---

**Input:** All parties hold the same set CONFLICTS and ZEROS. Each honest party not in CONFLICTS holds a pair of polynomials  $(f_i(x), g_i(y))$ , and it is guaranteed that all the shares of honest parties lie on the same bivariate polynomial  $S(x, y)$  with degree at most  $3t/2$  in  $x$  and  $t + d$  in  $y$ .

**The protocol:**

1. Every party sets HAVE-SHARES =  $[n] \setminus (\text{ZEROS} \cup \text{CONFLICTS})$ .
  2. For every  $j \in \text{CONFLICTS}$ :
    - (a) Each party  $P_i$  for  $i \in \text{HAVE-SHARES}$  sends  $(i, f_i(j))$  to  $P_j$ .
    - (b) Let  $(i, u_i)$  be the value  $P_j$  received from  $P_i$ . Moreover, for every  $i \in \text{ZEROS}$ , consider  $(i, u_i)$  with  $u_i = 0$ . Given all  $(i, u_i)_{i \notin \text{CONFLICTS}}$ ,  $P_j$  looks for a codeword of a polynomial of degree  $t + d$  with a distance of at most  $t/2$  from all the values it received (see Theorem 3.1). If there is such codeword, set  $g_j(y)$  to be the unique Reed-Solomon reconstruction. If there is no such a unique codeword, then  $P_j$  broadcasts  $\text{complaint}(j)$  and every party  $P_i$  for  $i \in \text{HAVE-SHARES}$  broadcasts  $\text{reveal}(i, j, f_i(j))$ .
  3. The dealer sets  $\text{Bad} = \emptyset$ . For each  $\text{reveal}(i, j, u)$  message broadcasted, the dealer verifies that  $u = f_i(j)$ . If not, then it adds  $i$  to  $\text{Bad}$ . The dealer broadcasts  $\text{Bad}$ .
  4. The parties go to Step 6a if one of the following is not true: (i)  $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$ ; (ii)  $\text{Bad} \subset \text{HAVE-SHARES}$ . The parties go to Step 6b if  $|\text{Bad}| > t/2$ .
  5. Otherwise, for every  $j \in \text{CONFLICTS}$ , if  $\text{complaint}(j)$  was broadcasted, then the parties consider all the points  $R_j = \{(i, u_i)\}$  such that  $\text{reveal}(i, j, u_i)$  was broadcasted in Step 2b, and  $i \in \text{HAVE-SHARES} \setminus \text{Bad}$ , or  $u_i = 0$  if  $i \in \text{ZEROS}$ . They verify if  $R_j$  defines a unique polynomial of degree  $t + d$ . If not, they go to Step 6a. Otherwise,  $P_j$  sets  $g_j(y)$  to be that unique polynomial.
  6. **Output:**
    - (a) Discard the dealer: Output  $\text{discard}$ .
    - (b) Detect: Output  $(\text{detect}, \text{Bad})$ .
    - (c) Proceed: Each party  $j \in \text{CONFLICTS}$  outputs  $(\text{proceed}, \perp, g_j(y))$ . All other parties  $P_j$  with  $j \notin \text{CONFLICTS}$  output  $(\text{proceed}, f_j(x), g_j(y))$ .
- 

**Lemma 4.6.** *Protocol 4.5,  $\Pi_{\text{rec-g}}$ , perfectly securely computes Functionality 4.4,  $\mathcal{F}_{\text{rec-g}}$ , in the presence of a malicious adversary, controlling at most  $t < n/3$ . The protocol requires the transmission of  $\mathcal{O}(n^2 \log n)$  bits over point-to-point channels, and each party broadcasts at most  $\mathcal{O}(n \log n)$  bits.*

### 4.3 Reconstruction of $f$ -polynomials in CONFLICTS

The goal of this step is to make each party in CONFLICTS to receive its  $f$ -share. This is performed in a similar manner to that of reconstruction of  $g$ . This time, all honest parties hold shares of  $g$ , and thus each party in CONFLICTS receives at least  $2t + 1$  correct values on each its  $f$  polynomial. The  $f$ -polynomial is of degree  $3t/2$ , and therefore we fail to reconstruct if the adversary introduces more than  $t/2$  errors. In that case, we will have detection, in a similar manner to the reconstruction of  $g$ . The full details of the functionality (denoted by  $\mathcal{F}_{\text{rec-f}}$ ), and the protocol (denoted by  $\Pi_{\text{rec-f}}$ ), as well as the proof of the following lemma are given in the full version.

**Lemma 4.7.** *The Protocol  $\Pi_{\text{rec-f}}$ , perfectly securely computes the  $\mathcal{F}_{\text{rec-f}}$  functionality, in the presence of a malicious adversary, controlling at most  $t < n/3$ .*

### 4.4 Putting Everything Together: Packed Secret Sharing

We view a list of  $(t/2 + 1)(d + 1)$  secrets SECRETS as a  $(t/2 + 1) \times (d + 1)$  matrix.

---

**Functionality 4.8: Packed Secret Sharing** –  $\mathcal{F}_{\text{PSS}}$

---

The functionality is parameterized by the set of corrupted parties  $I \subseteq [n]$ .

- **Input:** All parties input a set  $\text{ZEROS} \subseteq [n]$  such that  $|\text{ZEROS}| \leq t$ . If the dealer is honest then it is guaranteed that  $\text{ZEROS} \subseteq I$ .
  - **Honest dealer:** The dealer sends SECRETS to  $\mathcal{F}_{\text{PSS}}$ . The functionality sends ZEROS to the adversary, which replies with  $(f_i(x), g_i(y))_{i \in I}$  under the constraint that  $f_i(x) = g_i(y) = 0$  for every  $i \in \text{ZEROS}$ . The functionality chooses a random bivariate polynomial  $S(x, y)$  of degree  $3t/2$  in  $x$  and  $t + d$  in  $y$  under the constraints that (i) SECRETS is embedded in  $S$  (see Section 3 for the meaning of embedding); (ii)  $S(x, i) = f_i(x)$  for every  $i \in I$ ; (iii)  $S(i, y) = g_i(y)$ .
  - **Corrupted dealer:** The functionality sends ZEROS to the adversary, which replies with  $S(x, y)$ .  $\mathcal{F}_{\text{PSS}}$  verifies that  $S(x, y)$  is of degree  $3t/2$  in  $x$  and degree  $t + d$  in  $y$ , and that for every  $i \in \text{ZEROS}$  it holds that  $f_i(x) = g_i(y) = 0$ . If not,  $\mathcal{F}_{\text{PSS}}$  replaces  $S(x, y) = \perp$ .
  - **Output:**  $\mathcal{F}_{\text{PSS}}$  sends to each party  $P_j$  the pair of polynomials  $S(x, j), S(j, y)$ .
- 

We claim that there is always a bivariate polynomial that can be reconstructed. Specifically, consider for simplicity the case where  $|I| = t$ :

1. A bivariate polynomial of degree  $3t/2$  in  $x$  and degree  $t + d$  in  $y$  is determined by  $(3t/2 + 1)(t + d + 1)$  values.
2. The adversary sends  $t$  pairs of polynomials of degree  $3t/2$  and  $t + d$ . The  $f$  polynomials define  $t(3t/2 + 1)$  values. Each  $g$  polynomial is already determined in  $t$  coordinates, and therefore we have a total of  $t(t + d + 1 - t) = t(d + 1)$ .
3. SECRETS determines  $(t/2 + 1) \cdot (d + 1)$  values.

Therefore, the number of constraints that we have is  $(t/2 + 1)(d + 1) + t(3t/2 + 1) + t(d + 1)$ , which is exactly  $(3t/2 + 1)(t + d + 1)$ , the total number of variables in the bivariate polynomial.

---

**Protocol 4.9: Packed Secret Sharing in the  $(\mathcal{F}_{\text{ShareAttempt}}, \mathcal{F}_{\text{rec-g}}, \mathcal{F}_{\text{rec-f}})$ -hybrid model  $-\Pi_{\text{PSS}}$** 


---

**Input:** The dealer holds SECRETS. All honest parties hold the same set ZEROS.

**The protocol:**

1. **Dealing the shares:**
    - (a) The dealer chooses a random bivariate polynomial  $S(x, y)$  of degree at most  $3t/2$  in  $x$  and degree  $t + d$  in  $y$  that embeds SECRETS, under the constraint that for every  $i \in \text{ZEROS}$  it holds that  $S(x, i) = 0$  and  $S(i, y) = 0$ .
    - (b) All parties invoke Functionality 4.1,  $\mathcal{F}_{\text{ShareAttempt}}$ , where the dealer inputs  $S(x, y)$  and all parties input ZEROS:
      - i. If the output is **discard**, then proceed to Step 4a.
      - ii. If the output is (**detect**, CONFLICTS) then set  $\text{ZEROS} = \text{ZEROS} \cup \text{CONFLICTS}$ . If  $|\text{ZEROS}| > t$  then proceed to Step 4a. Otherwise, go back to Step 1a.
      - iii. If the output is (**proceed**,  $f_i(x), g_i(y)$ , CONFLICTS), then proceed to the next step. Note that it must hold that (a) for parties  $i \in \text{CONFLICTS}$ ,  $f_i(x) = g_i(y) = \perp$  and (b)  $n - |\text{CONFLICTS}| \geq n - (t/2 - d)$ .
  2. **Reconstruct the  $g$ -polynomials:** The parties invoke Functionality 4.1,  $\mathcal{F}_{\text{rec-g}}$ , where each party  $P_i$  inputs (ZEROS, CONFLICTS,  $f_i(x), g_i(y)$ ).
    - (a) If the output is **discard**, then proceed to Step 4a.
    - (b) If the output is (**detect**, Bad) then set  $\text{ZEROS} = \text{ZEROS} \cup \text{Bad}$ . If  $|\text{ZEROS}| > t$  then discard and proceed to Step 4a. Otherwise, go back to Step 1a.
    - (c) Otherwise, the output is (**proceed**,  $f_i(x), g_i(y)$ ) where every party  $P_i$  with  $i \in \text{CONFLICTS}$  has  $g_i(y) \neq \perp$ , then proceed to the next step.
  3. **Reconstruct the  $f$ -polynomials:** The parties invoke Functionality  $\mathcal{F}_{\text{rec-f}}$ , where each party  $P_i$  inputs (ZEROS, CONFLICTS,  $f_i(x), g_i(y)$ ). Note that for parties in CONFLICTS it holds that  $f_i(x) = \perp$ .
    - (a) If the output of the functionality is **discard**, then proceed to Step 4a.
    - (b) If the output is (**detect**, Bad) then set  $\text{ZEROS} = \text{ZEROS} \cup \text{Bad}$ . If  $|\text{ZEROS}| > t$  then discard and go to Step 4a. Otherwise, go back to Step 1a.
    - (c) Otherwise, let (**proceed**,  $f_i(x), g_i(y)$ ) be the output, where now all parties have  $f_i(x) \neq \perp$  and  $g_i(y) \neq \perp$ . Go to Step 4b.
  4. **Output:**
    - (a) **Discard:** All parties output  $\perp$ .
    - (b) **Successful:** Output  $f_i(x), g_i(y)$ .
- 

**Lemma 4.10.** *Let  $t < n/3$  and  $d \leq t/4$ . Protocol 4.9,  $\Pi_{\text{PSS}}$ , perfectly securely computes Functionality 4.8,  $\mathcal{F}_{\text{PSS}}$ , in the  $(\mathcal{F}_{\text{ShareAttempt}}, \mathcal{F}_{\text{rec-g}}, \mathcal{F}_{\text{rec-f}})$ -hybrid model, in the presence of a malicious adversary, controlling at most  $t < n/3$ .*

**Communication and Efficiency Analysis.** We conclude the following lemma, proven in the full version:

**Lemma 4.11.** *Let  $t < n/3$  and  $d \leq t/4$ . There exists a protocol that implements Functionality 4.8, has a communication complexity of  $\mathcal{O}(n^2 \log n)$  bits over point-to-point channels and  $\mathcal{O}(n^2 \log n)$  bits broadcast for sharing  $\mathcal{O}((d + 1)n)$  values (i.e.,  $\mathcal{O}(n(d + 1) \log n)$  bits) simultaneously in  $\mathcal{O}(1)$  rounds. Every party broadcasts at most  $\mathcal{O}(n \log n)$  bits.*

## 5 Batched and Packed Secret Sharing

In this section, we suggest how to keep the broadcast unchanged when running  $m$  instances of the packed secret sharing with the same dealer. That is, if one



instance requires  $\mathcal{O}(n^2 \log n)$  bits communicated over point-to-point channels and each party (including the dealer) broadcasts  $\mathcal{O}(n \log n)$  bits, we have a protocol that requires  $\mathcal{O}(mn^2 \log n)$  bits communicated over point-to-point channels and each party still has to broadcast at most  $\mathcal{O}(n \log n)$  bits (and a total of  $\mathcal{O}(n^2 \log n)$ ). We review the changes necessary for each one of the sub-protocols of packed secret sharing.

**Sharing attempt and Batched Complaints.** Here the dealer inputs  $m$  bivariate polynomials, but there is *one* set  $\text{ZEROS} \subset [n]$ . It is assumed that all bivariate polynomials have 0 shares for the parties in  $\text{ZEROS}$ .

At Step 2b in Protocol 4.2, every  $P_i$  checks consistency in all instances but raises a complaint for only one of them, say, the minimum index of the instance. A complaint now looks like  $\text{complaint}(i, j, f_i(j), g_i(j), \alpha)$  where  $\alpha \in \{1, \dots, m\}$ . Moreover, if a party broadcasts  $\text{complaint}(i, j, u_i, v_i)$  for  $j \in \text{ZEROS}$ , then the dealer must add  $P_i$  to  $\text{CONFLICTS}$ . Thus, there is no need for  $P_i$  to broadcast such a complaint in each instance that it sees inconsistency with  $P_j$  for  $j \in \text{ZEROS}$ , but it is enough to do it in only one of the instances.

This keeps the broadcast cost  $\mathcal{O}(n^2 \log n)$  bits among all  $m$  instances combined (as opposed  $\mathcal{O}(mn^2 \log n)$  when running them simultaneously in a black-box manner).

Note that when the dealer is honest, honest parties never complain on one another, and this holds in all  $m$  invocations. Moreover, if the dealer is corrupted and two honest parties have to file a joint complaint, then both will have the exact same minimal index, and the dealer must have to add one of them into  $\text{CONFLICTS}$ , exactly as we have in single instance.

**Batched reconstruction of  $g$  polynomials in  $\text{CONFLICTS}$ .** Here the change in the protocol is more delicate than the previous case, and we provide a full modeling and proof. Specifically, In Step 2b of Protocol 4.2, a party  $P_j$  may fail to reconstruct  $g_j$  in multiple instances. However, it is enough to pick one instance  $\beta$  (say, the one with minimum index) and complains publicly with  $\beta$ . Now, rest of the public verification happens with respect to  $\beta$ th invocation. If parties publicly reveal values that are different than what they revealed privately, then the party knows that those parties are corrupted and can try to reconstruct the polynomials without those shares. In particular, the only case when a party cannot uniquely reconstruct is when the the adversary introduces more than  $t/2$  errors. However, if the public reconstruction of  $g$  in the  $\beta$ th execution is successful, it can recognize  $t/2$  misbehaving parties by comparing the polynomial that was publicly reconstruct to the shares sent to it privately. Note that it is possible that a corrupted party sends some share to  $P_j$  privately but makes some other value public.  $P_j$  knows for sure that such party is corrupt, even though  $\text{Bad}$  that the dealer broadcasts can even be empty. Once  $P_j$  recognizes more than  $t/2$  errors, it can eliminate them in all other private reconstructions, remaining with less than  $t/2$  errors in *all* the  $m$  executions. The functionality (denoted as  $\mathcal{F}_{\text{rec-g}}^{\text{batched}}$ ) and the full specification of the protocol (denoted as  $\Pi_{\text{rec-g}}^{\text{batched}}$ ) are given in the full version, as well as the proof of the following lemma:

**Lemma 5.1.** *Protocol  $\Pi_{\text{rec-g}}^{\text{batched}}$ , perfectly-securely computes  $\mathcal{F}_{\text{rec-g}}^{\text{batched}}$  in the presence of a malicious adversary, controlling at most  $t < n/3$ .*

**Batched reconstruction of  $f$ -polynomials in CONFLICTS.** This follows the exact same lines as the reconstruction of  $g$  polynomials. Specifically, if the local reconstruction is not unique, then it is enough to pick one instance  $\gamma \in [m]$  and open it publicly. The public verification happens with respect to the  $\gamma$ th instance.  $P_j$  will then be able to reconstruct  $f_j^\ell$  for every  $\ell \in [m]$ .

### 5.1 Sharing

To conclude, we realize the following functionality putting together the batched version of protocols for the sharing attempt, reconstruction of  $g$  and  $f$  polynomials. Referring the protocol as  $\Pi_{\text{PSS}}^{\text{batched}}$ , we culminate at the following theorem.

---

**Functionality 5.2: Batched and Packed Secret Sharing –  $\mathcal{F}_{\text{PSS}}^{\text{batched}}$**

---

The functionality is parameterized by the set of corrupted parties  $I \subseteq [n]$ .

- **Input:** All parties input a set  $\text{ZEROS} \subseteq [n]$  such that  $|\text{ZEROS}| \leq t$ . If the dealer is honest then it is guaranteed that  $\text{ZEROS} \subseteq I$ .
  - **Honest dealer:** The dealer sends  $(\text{SECRETS}_\ell)_{\ell \in [m]}$  to  $\mathcal{F}_{\text{PSS}}^{\text{batched}}$ . The functionality sends  $\text{ZEROS}$  to the adversary, who sends back  $(f_i^\ell(x), g_i^\ell(y))_{i \in I, \ell \in [m]}$  such that  $f_i^\ell(k) = g_k^\ell(i)$  for every  $i, k \in I$  and  $\ell \in [m]$ . Moreover, for every  $i \in \text{ZEROS}$ ,  $f_i(x) = g_i(y) = 0$ . For every  $\ell \in [m]$ , the functionality chooses a random bivariate polynomial  $S^\ell(x, y)$  of degree  $3t/2$  in  $x$  and  $t + d$  in  $y$  under the constraints that (i)  $\text{SECRETS}_\ell$  is embedded in  $S_\ell$ ; (ii)  $S^\ell(x, i) = f_i^\ell(x)$  for every  $i \in I$ ; (iii)  $S^\ell(i, y) = g_i^\ell(y)$ .
  - **Corrupted dealer:** For every  $\ell \in [m]$ , the dealer sends  $S^\ell(x, y)$  to  $\mathcal{F}_{\text{PSS}}^{\text{batched}}$  that verifies that  $S^\ell(x, y)$  is of degree  $3t/2$  in  $x$  and degree  $t + d$  in  $y$ , and for every  $i \in \text{ZEROS}$  it holds that  $f_i(x) = g_i(y) = 0$ . If not,  $\mathcal{F}_{\text{PSS}}^{\text{batched}}$  replaces  $S^\ell(x, y) = \perp$ .
  - **Output:**  $\mathcal{F}_{\text{PSS}}^{\text{batched}}$  sends to each party  $P_j$  the polynomials  $(S^\ell(x, j), S^\ell(j, y))_{\ell \in [m]}$ .
- 

**Theorem 5.3.**  $\Pi_{\text{PSS}}^{\text{batched}}$  securely computes  $\mathcal{F}_{\text{PSS}}^{\text{batched}}$  (Functionality 5.2). It requires a communication complexity of  $\mathcal{O}(mn^2 \log n)$  bits over-point-to-point channels and  $\mathcal{O}(n^2 \log n)$  bits broadcast for sharing  $\mathcal{O}((d + 1)mn)$  values (i.e.,  $\mathcal{O}((d + 1)mn \log n)$  bits) simultaneously in  $\mathcal{O}(1)$  rounds. Each party broadcasts at most  $\mathcal{O}(n \log n)$  bits.

### 5.2 Reconstruction

We present the reconstruction protocols for our batched and packed secret sharing. As mentioned in the introduction, for our detectable secret sharing, we get a detectable reconstruction, a weaker form of robust reconstruction. For the case of  $d = 0$ , we get robust reconstruction, and so verifiable secret sharing. We start with fully specifying the functionality.

**Functionality 5.4: Detectable Reconstruction for Batched and Packed Secret Sharing** –  $\mathcal{F}_{\text{PSS-Rec}}^{\text{batched}}$

The functionality is parameterized with the set of corrupted parties  $I \subset [n]$ .

1. **Input:** All honest parties send  $\text{ZEROS} \subset [n]$ . When the dealer is honest,  $\text{ZEROS} \subseteq I$ . Each honest party  $P_j$  sends  $(f_j^k(x), g_j^k(y))$  for each  $k \in [m]$  and  $j \notin I$ . For each  $k$ , let  $S^k(x, y)$  be the unique bivariate polynomial of degree  $3t/2$  in  $x$  and  $t+d$  in  $y$  that satisfies  $f_j^k(x) = S^k(x, j)$  and  $g_j^k(y) = S^k(j, y)$  for every  $j \notin I$ .
2. Send  $\text{ZEROS}$  and  $S^1(x, y), \dots, S^m(x, y)$  to the adversary. If  $d = 0$  then go to Step 4c.
3. Receive back from the adversary a message  $M$ .
4. **Output:**
  - (a) If  $M = \text{discard}$  and the dealer is corrupted, then send  $\text{discard}$  to all parties.
  - (b) If  $M = (\text{detect}, \text{Bad})$  with  $|\text{Bad}| > t/2$  and  $\text{Bad} \cap \text{ZEROS} = \emptyset$ , and in case of an honest dealer  $\text{Bad} \subseteq I$ , then send  $(\text{detect}, \text{Bad})$  to all parties.
  - (c) If  $M = \text{proceed}$  then send to each  $j$  the output  $(\text{proceed}, S^1(x, y), \dots, S^m(x, y))$ .

Note that if the dealer is honest then  $\text{discard}$  cannot occur. Moreover, if the dealer is honest and  $|\text{ZEROS}| > t/2$ , the  $(\text{detect}, \text{Bad})$  cannot occur, as  $|\text{Bad} \cup \text{ZEROS}| \leq t$  and so we cannot have  $|\text{Bad}| > t/2$ . In that case, we always succeed to reconstruct. On the other hand, if the dealer is honest and  $|\text{ZEROS}| \leq t/2$ , the adversary might cause to a failure. In that case, we are guaranteed to have a mass detection.

**The protocol.** To reconstruct shared polynomials  $S^1(x, y), \dots, S^m(x, y)$ , the reconstruction protocol follows a similar structure of that of Protocol  $\Pi_{\text{rec-g}}^{\text{batched}}$ :

1. Each party  $P_i$  holds  $(f_i^\ell, g_i^\ell(y))_{\ell \in [m]}$  and a set  $\text{ZEROS} \subset [n]$ .
2. Each party now sends all its polynomials  $f_i^1(x), \dots, f_i^m(x)$  over the private channel to all other parties.
3. The parties try to reconstruct polynomials  $g_1^\ell(y), \dots, g_n^\ell(y)$  using the polynomials  $f_1^\ell(x), \dots, f_n^\ell(x)$  (and taking 0 for the parties in  $\text{ZEROS}$ ). E.g., reconstruct  $g_j^\ell(y)$  by considering  $(k, f_k^\ell(j))_{k \notin \text{ZEROS}}$  and adding  $(k, 0)$  for  $k \in \text{ZEROS}$ . Try to correct at most  $t/2$  errors, for every  $\ell \in [m]$  (see Theorem 3.1). If some party fails to decode some polynomial  $g_j^\ell(y)$ , then it broadcast  $\text{complaint}(j, \ell)$ . Note that it is enough to broadcast just a single complaint, say the one with the lexicographically smallest  $j, \ell$ .
4. We will have a public reconstruction of  $g_j^\ell(y)$ : Each party broadcasts its point on that polynomial, and the dealer broadcasts a set  $\text{Bad}$  if there are any wrong values broadcasted. The parties output  $(\text{detect}, \text{Bad})$  if  $|\text{Bad}| > t/2$ . The parties check that when excluding all points in  $\text{Bad}$  then all points lie on a single polynomial  $g_j^\ell(y)$ .
5. Using the public reconstruction, the party  $P_j$  can now locate  $t/2$  corruptions and reconstruct (see full version) all polynomials  $g_1^\ell(y), \dots, g_n^\ell(y)$  for every  $\ell \in [m]$ . All parties can now find unique bivariate polynomials  $S^\ell(x, y)$  satisfying  $S^\ell(i, y) = g_i^\ell(y)$  for every  $i \in [n]$ . The parties output those polynomials.

There are few properties that we would like to highlight with respect to the above protocol:

1. Note that when  $d = 0$ , then we can simply run Reed-Solomon decoding in Step 3 and always succeed to reconstruct as Reed Solomon decoding returns unique decoding when there are at most  $t$  errors. Thus, there is no need for public resolution.
2. There are at most  $n$  complaints, which lead to each party broadcasting at most  $\mathcal{O}(n \log n)$  bits to resolve all complaints.

**Conclusion: Detectable Secret Sharing.** While we provide functionality-based modeling and proofs, the verifiable secret sharing literature is also full of property based definitions, and some readers might find such modeling helpful. We provide here such properties for completeness. From combining Functionalities 5.2 and 5.4, when using  $d > 0$  we obtain a two-phase protocol for parties  $\mathcal{P} = \{P_1, \dots, P_n\}$  where a distinguished dealer  $P^* \in \mathcal{P}$  holds initial SECRETS, and all honest parties hold the same set  $\text{ZEROS}_{P^*} \subseteq [n]$  (where no honest party is in  $\text{ZEROS}_{P^*}$  if  $P^*$  is honest) such that the following properties hold:

- **Secrecy:** If the dealer is honest during the first phase (the sharing phase), then at the end of this phase, the joint view of the malicious parties is independent of the dealer’s input SECRETS.
- **Reconstruction or detection – corrupted dealer:** At the end of the sharing phase, the joint view of the honest parties define values SECRETS’ such that at the end of the reconstruction phase – all honest parties will output either SECRETS’, or discard the dealer, or  $t/2$  new values will be added to  $\text{ZEROS}_{P^*}$ .
- **Reconstruction or detection – honest dealer:** At the end of the sharing phase, the joint view of the honest parties define values SECRETS’ = SECRETS that the dealer used as input for the sharing phase. At the end of the reconstruction phase, all honest parties will output SECRETS, or  $t/2$  new indices, all of corrupted parties, will be added to  $\text{ZEROS}_{P^*}$ . If  $\text{ZEROS}_{P^*}$  initially contained more than  $t/2$  values during the sharing phase, then the output of the second phase is always SECRETS.

When  $|\text{SECRETS}| \in \Omega(n^2)$ , the protocol uses  $\mathcal{O}(n^4 \log n + |\text{SECRETS}| \log n)$  communication complexity for both sharing and reconstruction.

**Conclusion: Verifiable Secret Sharing.** From combining Functionalities 5.4 and 5.4, when using  $d = 0$  we obtain a verifiable secret sharing: A two-phase protocol for parties  $\mathcal{P} = \{P_1, \dots, P_n\}$  where a distinguished dealer  $P^* \in \mathcal{P}$  holds initial secrets  $s_1, \dots, s_t$  is a Verifiable Secret Sharing Protocol tolerating  $t$  malicious parties and the following conditions hold for any adversary controlling at most  $t$  parties:

- **Validity:** Each honest party  $P_i$  outputs the values  $s_{i,1}, \dots, s_{i,t}$  at the end of the second phase (the reconstruction phase). Furthermore, if the dealer is honest then  $(s_{i,1}, \dots, s_{i,t}) = (s_1, \dots, s_t)$ .
- **Secrecy:** If the dealer is honest during the first phase (the sharing phase) then at the end of this phase, the joint view of the malicious parties is independent of the dealer’s input  $s_1, \dots, s_t$ .

- **Reconstruction:** At the end of the sharing phase, the joint view of the honest parties defines values  $s'_1, \dots, s'_t$  such that all honest parties will output  $s'_1, \dots, s'_t$  at the end of the reconstruction phase.

When  $|\text{SECRETS}| \in \Omega(n)$ , the protocol uses  $\mathcal{O}(n^4 \log n + |\text{SECRETS}| \cdot n \log n)$  communication complexity for both sharing and reconstruction.

## 6 Packed and Batched Verifiable Triple Sharing

Packed verifiable triple sharing (VTS) allows a dealer to verifiably share  $t/2 + 1$  multiplication triples at the cost of incurring  $\mathcal{O}(n^2)$  elements of communication over point-to-point channels as well as broadcast. Precisely, VTS outputs each element of the triples to be Shamir-shared via a degree- $t$  polynomial. In the full version, we show how to implement such packed verifiable triple sharing. We will also present the batched version, denoted as  $\Pi_{\text{PVTS}}^{\text{batched}}$ , where  $\mathcal{O}(mn)$  shared triplets are prepared with  $\mathcal{O}(mn^2)$  elements of communication over point-to-point channels and the same broadcast as needed for one instance (i.e.  $\mathcal{O}(n^2)$ ). This is an important contribution of this work that utilizes our both verifiable secret sharing and detectable secret sharing constructions, and we refer the reader to the full version for further details.

## 7 The MPC Protocol

We now describe our complete MPC protocol as a composition of the building blocks, the PSS (Sects. 4, 5) and the VTS (Sect. 6) protocols, as well as other building blocks from the literature (e.g., triplets extractor ( $\Pi_{\text{tripleExt}}$ ) and batch Beaver multiplication ( $\Pi_{\text{bBeaver}}$ ), see full version). The protocol  $\Pi_{\text{MPC}}$  and the corresponding functionality  $\mathcal{F}_{\text{MPC}}$  are provided below. At a high level, the protocol is divided into the following two phases:

1. *Beaver triple generation:* In this phase, parties generate  $C$  number of degree- $t$  Shamir-shared multiplication triples where,  $C$  denotes the number of multiplication gates in the circuit. Towards that, each party first generates triples using our VTS protocol. Subsequently, a triple extraction protocol “merges” the triples generated by all parties and “extracts” random triples (not known to any party) which will be consumed in the second phase. For sufficiently large circuits, specifically for circuits of size  $\Omega(n^3)$ , this phase incurs an amortized cost of  $\mathcal{O}(n \log n)$  bits point-to-point communication per triple.
2. *Circuit computation:* Upon sharing of inputs by the input holding parties, in this phase the computation of the circuit proceeds by parties performing shared evaluation of the circuit. Since our sharing is linear, the linear operations of addition and multiplication by a constant are local. For multiplication of shared values, parties consume the Beaver triples generated in the prior phase. This is followed by the reconstruction of the outputs to the designated parties to complete the circuit evaluation.

**Functionality 7.1: MPC –  $\mathcal{F}_{\text{MPC}}$**

**Input:** Each  $P_i$  holds input  $x_i \in \mathbb{F} \cup \{\perp\}$ .

**Common Input:** An  $n$ -party function  $f(x_1, \dots, x_n)$ .

1. Each  $P_i$  sends  $x_i$  to the functionality. For any  $P_i$ , if  $x_i$  is outside the domain or  $P_i$  did not send any input, set  $x_i$  to a predetermined default value.
2. Compute  $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$  and send  $y_i$  to  $P_i$  for every  $i \in [n]$ .

**Protocol 7.2: MPC –  $\Pi_{\text{MPC}}$**

**Common input:** The description of a circuit, the field  $\mathbb{F}$ ,  $n$  non-zero distinct elements  $1, \dots, n$  and a parameter  $h$  where  $n = 2h + 1$ . Let  $m = \lceil \frac{C}{h+1-t} \rceil$ .

**Input:** Parties hold their inputs (belonging to  $\mathbb{F} \cup \{\perp\}$ ) to the circuit.

**(Beaver triple generation:)**

1. Each  $P_i$  chooses  $m + n(t/2 + 1)$  random multiplication triples and executes  $\Pi_{\text{PVTS}}^{\text{batched}}$  (Section 6) batching  $\lceil \frac{m}{(t/2+1)} \rceil + n$  instances each with  $t/2 + 1$  triples. Let  $(\langle a_i^j \rangle, \langle b_i^j \rangle, \langle c_i^j \rangle)$  for  $j \in [m]$  denote the triples shared by  $P_i$ .
2. Parties execute  $m$  instances of  $\Pi_{\text{tripleExt}}$  with  $(\langle a_i^j \rangle, \langle b_i^j \rangle, \langle c_i^j \rangle)$  for every  $i \in [n]$  as the input for the  $j^{\text{th}}$  instance. Let  $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle)$  for  $i \in [C]$  denote the random multiplication triples generated.

**(Circuit computation:)**

1. **(Input)** Each party  $P_i$  holding  $k_i$  inputs to the circuit executes  $\Pi_{\text{PSS}}^{\text{batched}}$  (Section 5) batching  $\lceil \frac{k_i}{t/2+1} \rceil$  instances to share its inputs.
2. **(Linear Gates)** Parties locally apply the linear operation on their respective shares of the inputs.
3. **(Multiplication Gates)** Let  $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle)$  be the multiplication triple associated with the  $i^{\text{th}}$  multiplication gate with shared inputs  $(\langle x_i \rangle, \langle y_i \rangle)$ . Parties invoke  $\Pi_{\text{bBeaver}}$  with  $\{\langle x_i \rangle, \langle y_i \rangle, \langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle\}$  for all gates  $i$  at the same layer of the circuit and obtain the corresponding  $\langle z_i \rangle$  as the output sharing for every gate  $i$ .
4. **(Output)** For each output gate  $j$  with the associated sharing  $\langle v_j \rangle$ , parties execute  $\Pi_{\text{Rec}}$  towards every party  $P_i$  who is supposed to receive the output  $v_j$ .

**Theorem 7.3.** *Let  $t < n/3$ . Protocol 7.2 securely implements  $\mathcal{F}_{\text{MPC}}$  (Functionality 7.1) and has a communication complexity of  $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$  bits over point to point channels and  $\mathcal{O}(n^3 \log n)$  bits broadcast for evaluating a circuit with  $C$  gates and depth  $D$  in expected  $\mathcal{O}(D)$  rounds. Every party broadcasts  $\mathcal{O}(n^2 \log n)$  bits.*

**Acknowledgements.** Gilad Asharov is sponsored by the Israel Science Foundation (grant No. 2439/20), by JPM Faculty Research Award, and by the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 891234. Shravani Patil would like to acknowledge the support of DST National Mission on Interdisciplinary Cyber-Physical Systems (NM-ICPS) 2020–2025. Arpita Patra would like to acknowledge the support of DST National Mission on Interdisciplinary Cyber-Physical Systems (NM-ICPS) 2020–2025, Google India Faculty Award, and JPM Faculty Research Award.

## References

1. Abraham, I., Asharov, G., Patil, S., Patra, A.: Asymptotically free broadcast in constant expected time via packed vss. In: TCC (2022). [https://doi.org/10.1007/978-3-031-22318-1\\_14](https://doi.org/10.1007/978-3-031-22318-1_14)
2. Abraham, I., Asharov, G., Yanai, A.: Efficient perfectly secure computation with optimal resilience. In: Theory of Cryptography (2021). [https://doi.org/10.1007/978-3-030-90453-1\\_3](https://doi.org/10.1007/978-3-030-90453-1_3)
3. Abraham, I., Dolev, D., Halpern, J.Y.: An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. In: PODC 2008 (2008). <https://doi.org/10.1145/1400751.1400804>
4. Anirudh, C., Choudhury, A., Patra, A.: A survey on perfectly-secure verifiable secret-sharing. Cryptology ePrint Archive (2021)
5. Asharov, G., Cohen, R., Shochat, O.: Static vs. adaptive security in perfect MPC: a separation and the adaptive security of BGW. In: 3rd Conference on Information-Theoretic Cryptography, ITC 2022 (2022)
6. Asharov, G., Lindell, Y.: A full proof of the BGW protocol for perfectly secure multiparty computation. J. Cryptol. **30**(1), 58–151 (2015). <https://doi.org/10.1007/s00145-015-9214-4>
7. Asharov, G., Lindell, Y., Rabin, T.: Perfectly-secure multiplication for any  $t < n/3$ . In: Advances in Cryptology - CRYPTO 2011 (2011). [https://doi.org/10.1007/978-3-642-22792-9\\_14](https://doi.org/10.1007/978-3-642-22792-9_14)
8. Bangalore, L., Choudhury, A., Patra, A.: Almost-surely terminating asynchronous byzantine agreement revisited. In: 2018 ACM Symposium on Principles of Distributed Computing, PODC. ACM (2018). <https://doi.org/10.1145/3212734.3212735>
9. Bangalore, L., Choudhury, A., Patra, A.: The power of shunning: Efficient asynchronous byzantine agreement revisited\*. J. ACM (2020)
10. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Annual International Cryptology Conference (1991). [https://doi.org/10.1007/3-540-46766-1\\_34](https://doi.org/10.1007/3-540-46766-1_34)
11. Beerliova-Trubiniova, Z., Hirt, M.: Efficient multi-party computation with dispute control. In: Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4–7 2006, Proceedings 3, pp. 305–328 (2006). [https://doi.org/10.1007/11681878\\_16](https://doi.org/10.1007/11681878_16)
12. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure MPC with linear communication complexity. In: Theory of Cryptography Conference (2008). [https://doi.org/10.1007/978-3-540-78524-8\\_13](https://doi.org/10.1007/978-3-540-78524-8_13)
13. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Annual ACM Symposium on Theory of Computing (1988). <https://doi.org/10.1145/62212.62213>
14. Ben-Sasson, E., Fehr, S., Ostrovsky, R.: Near-linear unconditionally-secure multiparty computation with a dishonest minority. In: Advances in Cryptology-CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2012. Proceedings, pp. 663–680 (2012). [https://doi.org/10.1007/978-3-642-32009-5\\_39](https://doi.org/10.1007/978-3-642-32009-5_39)
15. Berman, P., Garay, J.A., Perry, K.J.: Bit optimal distributed consensus. In: Computer science (1992)

16. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: FOCS (2001). <https://doi.org/10.1109/SFCS.2001.959888>
17. Canetti, R., Damgaard, I., Dziembowski, S., Ishai, Y., Malkin, T.: On adaptive vs. non-adaptive security of multiparty protocols. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 262–279. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-44987-6\\_17](https://doi.org/10.1007/3-540-44987-6_17)
18. Canetti, R., Damgaard, I., Dziembowski, S., Ishai, Y., Malkin, T.: Adaptive versus non-adaptive security of multi-party protocols. *J. Cryptol.* **17**(3), 153–207 (2004). <https://doi.org/10.1007/s00145-004-0135-x>
19. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing (1988). <https://doi.org/10.1145/62212.62214>
20. Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In: 26th Annual Symposium on Foundations of Computer Science (1985). <https://doi.org/10.1109/SFCS.1985.64>
21. Choudhury, A.: Protocols for Reliable and Secure Message Transmission. Ph.D. thesis, Citeseer (2010)
22. Choudhury, A., Patra, A.: An efficient framework for unconditionally secure multiparty computation. *IEEE Trans. Inf. Theory.* **63**, 428–468 (2016)
23. Coan, B.A., Welch, J.L.: Modular construction of nearly optimal byzantine agreement protocols. In: ACM Symposium on Principles of distributed computing (1989). <https://doi.org/10.1145/72981.73002>
24. Cramer, R., Damgård, I., Maurer, U.: General secure multi-party computation from any linear secret-sharing scheme. In: International Conference on the Theory and Applications of Cryptographic Techniques (2000). [https://doi.org/10.1007/3-540-45539-6\\_22](https://doi.org/10.1007/3-540-45539-6_22)
25. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: 20th Annual ACM Symposium on Theory of Computing (1988). <https://doi.org/10.1145/62212.62225>
26. Feldman, P.N.: Optimal algorithms for Byzantine agreement. Ph.D. thesis, Massachusetts Institute of Technology (1988)
27. Fischer, M.J., Lynch, N.A.: A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.* (1982)
28. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: 24th Annual ACM Symposium on Theory of Computing (1992). <https://doi.org/10.1145/129712.129780>
29. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: ACM Symposium on Principles of Distributed Computing (1998). <https://doi.org/10.1145/277697.277716>
30. Goyal, V., Liu, Y., Song, Y.: Communication-efficient unconditional MPC with guaranteed output delivery. In: Annual International Cryptology Conference (2019). [https://doi.org/10.1007/978-3-030-26951-7\\_4](https://doi.org/10.1007/978-3-030-26951-7_4)
31. Goyal, V., Song, Y., Zhu, C.: Guaranteed output delivery comes free in honest majority MPC. In: Advances in Cryptology-CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II, pp. 618–646 (2020). [https://doi.org/10.1007/978-3-030-56880-1\\_22](https://doi.org/10.1007/978-3-030-56880-1_22)



32. Hirt, M., Maurer, U., Przydatek, B.: Efficient secure multi-party computation. In: International Conference on the Theory and Application of Cryptology and Information Security (2000). [https://doi.org/10.1007/3-540-44448-3\\_12](https://doi.org/10.1007/3-540-44448-3_12)
33. Katz, J., Koo, C.: On expected constant-round protocols for byzantine agreement. In: Annual International Cryptology Conference (2006). [https://doi.org/10.1007/11818175\\_27](https://doi.org/10.1007/11818175_27)
34. Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically secure protocols and security under composition. In: 38th Annual ACM Symposium on Theory of Computing (2006). <https://doi.org/10.1145/1132516.1132532>
35. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error Correcting Codes, vol. 16. Elsevier (1977)
36. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: ACM Symposium on Theory of Computing (1989). <https://doi.org/10.1145/73007.73014>



# An Incremental PoSW for General Weight Distributions

Hamza Abusalah<sup>1</sup>(✉) and Valerio Cini<sup>2</sup>

<sup>1</sup> IMDEA Software Institute, Madrid, Spain  
hamza.abusalah@imdea.org

<sup>2</sup> Austrian Institute of Technology, Vienna, Austria  
valerio.cini@ait.ac.at

**Abstract.** A proof of sequential work (PoSW) scheme allows the prover to convince a verifier that it computed a certain number of computational steps sequentially. Very recently, graph-labeling PoSW schemes, found applications in light-client blockchain protocols, most notably bootstrapping. A bootstrapping protocol allows a light client, with minimal information about the blockchain, to hold a commitment to its stable prefix.

An incremental PoSW (iPoSW) scheme allows the prover to non-trivially increment proofs: given  $\chi, \pi_1$  and integers  $N_1, N_2$  such that  $\pi_1$  is a valid proof for  $N_1$ , it generates a valid proof  $\pi$  for  $N_1 + N_2$ .

In this work, we construct an iPoSW scheme based on the skiplist-based PoSW scheme of Abusalah et al. and prove its security in the random oracle model by employing the powerful on-the-fly sampling technique of Döttling et al. Moreover, unlike the iPoSW scheme of Döttling et al., ours is the first iPoSW scheme which is suitable for constructing incremental non-interactive arguments of chain knowledge (SNACK) schemes, which are at the heart of space and time efficient blockchain light-client protocols. In particular, our scheme works for general weight distributions, which we characterize as incrementally sampleable distributions. Our general treatment recovers the distribution underlying the scheme of Döttling et al. as well as the distribution underlying SNACK-enabled bootstrapping application as special cases. In realizing our general construction, we develop a new on-the-fly sampling technique.

## 1 Introduction

Proofs of Work (PoW) was introduced by Dwork and Naor [8], and in the past years has become very popular in the context of cryptocurrencies. A PoW scheme allows a prover to convince a verifier that a certain amount of computation was performed. However, this says nothing about whether the computation was done sequentially or in parallel.

A Proof of Sequential Work [12] (PoSW) is an (interactive) proof system in which the prover, on common inputs an integer parameter  $N$  and a statement  $\chi$ , computes a proof that convinces the verifier that  $N$  *sequential* computational steps have been performed since  $\chi$  was received.

A simple PoSW scheme based on a random oracle  $\tau : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$  is a hash chain: the prover computes  $y_i := \tau(\chi, y_{i-1})$  for  $i \in [N] := \{1, \dots, N\}$  and  $y_0 := \tau(\chi)$  and defines its proof  $\pi := y_N$ . The verifier verifies  $\pi$  by recomputation. From basic properties of random oracles, any accepting proof must have been computed, with overwhelming probability, in  $N$  sequential steps even by a massively parallel adversarial prover. From the prover’s perspective, this scheme is optimal: the prover does exactly  $N$  sequential steps and keeps constant size ( $\lambda$  bits) memory during its computation. However, the verifier needs to recompute the proof, and hence, spend as much resources as the honest prover. Therefore, for the applicability of PoSW schemes, one would also require succinctness, beyond completeness and soundness. Succinctness requires that the proof size as well as the verifier’s running time are poly-logarithmic in  $N$ .

Beyond classical applications of PoSW schemes for time stamping, where a prover wants prove to future verifiers that it stamped a certain message some time in the past, we have now more applications in the blockchain arena. The first such application uses a special form of a PoSW scheme which has *unique* proofs. Chia [6], in an effort to designing sustainable blockchains, combines in its mining process proofs of space [1, 9] and verifiable delay functions [4], which are a subclass of PoSW with unique proofs. While generating these proofs requires large space and sequential time resources, they must be efficiently and publicly verifiable.

Recently, (graph-labeling) PoSW schemes found applications in light-client blockchain protocols, most notably *bootstrapping* [2]. A light-client, which has minimal information about the blockchain in question, say its genesis block, is said to have securely bootstrapped if it ends up, after potentially talking to multiple provers, holding a commitment to the honest stable-prefix of the blockchain. The light client must be efficient in the sense that its verification time is at most poly-logarithmic in the length of the blockchain.

In the above applications, different classes of PoSW schemes are assumed. When the sequential computation is used in mining<sup>1</sup> as in Chia, the PoSW is required to have *unique* proofs. This subclass of PoSW schemes is called verifiable delay functions VDFs [4, 10, 13, 14]. The subclass used for bootstrapping application is called *graph-labeling* PoSW schemes [2, 3, 5, 7, 12]. It is not known how to use VDFs to solve the bootstrapping problem, nor how to use graph-labeling PoSW to get *unique* proofs, which is what is needed in the mining application.

In this work, we focus on the class of graph-labeling PoSW schemes, however, the same question that we address here in terms of incrementality is relevant for VDFs; in fact these are called *continuous* VDFs [10]. One main motivation

---

<sup>1</sup> In Chia, the mining resource is space, and entities that dedicate space and produce PoSpace proofs are called farmers, while entities that *finalize* the PoSpace-mined blocks by producing a VDF computation are called time lords. While PoSpace farmers compete on their PoSpace proofs, a single time lord suffices for each block. It is important to notice that farmers don’t compute VDFs.

of making this choice is that GL-PoSW schemes can be used to provide efficient solutions to blockchain light-client bootstrapping [2].

### 1.1 Graph-Labeling PoSW Schemes

A Graph-Labeling PoSW scheme is a PoSW scheme for a weighted graph family  $\Gamma = ((G_N, \Omega_N))_{N \geq 0}$ , where  $G_N = ([N]_0, E_G)$  is a DAG on  $[N]_0 := \{0, \dots, N\}$  and  $\Omega_N : [N] \rightarrow [0, 1]$  is a weigh function, i.e.,  $\sum_{i=1}^N \Omega_N(i) = 1$ . We refer to weight functions as probability distribution when convenient. All existing GL-PoSW schemes [2, 3, 5, 7, 12] are in the random oracle model. The prover, upon receiving a statement  $\chi$  from the verifier, uses  $\chi$  to refresh a random oracle  $\tau : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$  to compute a labeling  $L : [N]_0 \rightarrow \{0, 1\}^\lambda$  of  $G_N$ , where the label of  $i \in [N]_0$  is simply defined as  $L(i) := \tau(\chi, i, L(\text{Parents}(i)))$ , where  $\text{Parents}(i)$  denotes the (ordered) set of nodes with outgoing edges directed towards  $i$ . The prover then sends to the verifier a (vector) commitment  $\phi$  to the labeling  $L$ . The verifier then engages with the prover in a challenge response protocol in which for each challenge  $i \in [N]$  drawn by the verifier according to distribution induced by  $\Omega_N$ , the prover responds with protocol- and challenge-specific labels and openings from  $L$ , which the verifier then checks for consistency. (A part of the prover's response to challenge  $i$  is a  $\phi$ -opening at position  $i$ .)

Completeness,  $(\alpha, \epsilon)$ -soundness, and succinctness are required. Completeness stipulates that the verifier always accepts honest proofs, and succinctness requires that for any honestly generated proof  $\pi$ , it holds that  $|\pi| \in \text{poly}(\lambda, \log N)$ , and the running time of the verifier is upper-bonded by  $\text{poly}(\lambda, \log N)$ , where  $\lambda$  is a security parameter. Now  $(\alpha, \epsilon)$ -soundness guarantees that except with probability  $\epsilon$ , the verifier rejects any proof generated by a prover which made less than  $\alpha N$  sequential queries to  $\tau(\cdot)$ .

Such an interactive protocol is then made non-interactive in the ROM by applying the Fiat-Shamir [11] transform. The GL-PoSW scheme of [7] is defined and constructed non-interactively, but otherwise it follows the above design template.

The first PoSW scheme [12] is a GL-PoSW whose underlying graph  $G_N$  is *depth robust*, has the disadvantage of large space requirement  $\approx \lambda N$  bits during the prover's computation.

The high space requirement was addressed in [5], where they design a simple tree-based DAG  $G_N$ , which when used in the above design template, allows for a practically efficient GL-PoSW, where the prover requires much less space  $\ll \lambda N$  bits. Moreover, they show general space-time tradeoffs of the prover strategy, that we will discuss shortly.

In [3], another practically efficient GL-PoSW scheme that uses a skiplist DAG as its underlying graph was proposed. The scheme uses a slightly different, but essentially the same labeling strategy: instead of defining  $L(i) := \tau(\chi, i, L(\text{Parents}(i)))$ , it is computed using a permutation whose input is selected among  $L(\text{Parents}(i))$ . The permutation-based labeling gives the PoSW the additional feature of *reversibility*: given  $L(i)$  of any  $i \in [N]$ , one can efficiently

compute  $L(i - 1)$ . Reversibility comes at the cost of having labels of larger size: for parameter  $N$ , it holds that  $|L(i)| = \lambda \cdot \log N$ .

Motivated by applications of GL-PoSW schemes in designing light-client blockchain applications, a variant of the skiplist PoSW scheme is proposed in [2]. This scheme is identical to the original skiplist construction, except that labels are defined as  $L(i) := \tau(\chi, i, L(\text{Parents}(i)))$  for a random oracle  $\tau$ . The resulting scheme is no longer reversible as in [3]. However, it has the advantage of smaller labels  $|L(i)| = \lambda$ . Having constant size labels is important in the design of blockchains which allow for light-client protocols, where in the  $i$ th block of the respective blockchain, a PoSW label  $L(i)$  is stored, and having a constant size label for an ever-growing blockchain saves a lot of otherwise wasted storage. (For more details on such augmented blockchains, see [2].)

For simplicity, we refer to the scheme from [5] as the tree construction TC, and to both schemes from [3] and [2] as the skiplist construction SC.

*Space-Efficient Provers.* In all these schemes, a prover sequentially computing the labeling  $L$  of the PoSW underlying graph  $G_N$ , which would take  $N$  sequential invocations to its random oracle  $\tau$ , and storing all labels  $L$ , would be able to compile a convincing proof  $\pi$  out of  $L$ . However, if the prover does actually spend  $N$  sequential computations in computing  $L$  but stored nothing beyond  $L(N)$ , then it may need to spend another  $N$  sequential queries to be able to compile a convincing proof  $\pi$ . This issue can be seen as either a space-efficiency or soundness slack issue: while honest provers with essentially no storage need to spend  $2N$  steps, soundness is quantified over malicious provers doing  $< \alpha N$  sequential computation for  $\alpha \in (0, 1]$

This space issue was raised in both [12] and [5], where in the former it was left open, and in the latter, a general space-time tradeoff of the prover’s strategy was given. Roughly speaking, for space  $\approx \sqrt{N}$  an honest prover can spend  $\approx N + \sqrt{N}$  and convince the verifier that it did  $N$  sequential steps.

## 1.2 Incremental PoSW

This gap between the number of sequential steps of the honest and malicious provers, was essentially closed by Döttling, Lai and Malavolta [7], who give a variant of TC PoSW, call it ITC, in which the prover uses space up to  $\text{poly}(\lambda, \log N)$  and spends  $N$  sequential steps to convince the verifier that it did  $N$  sequential steps. In doing so, they introduce the notion of an *incremental* PoSW. This is a *non-interactive* GL-PoSW scheme  $(\text{P}, \text{V}, \text{Inc})$ , in which, in addition to the prover/verifier algorithms, there is an additional  $\text{Inc}$  algorithm that given  $\chi, \pi_1$  and integers  $N_1, N_2$  such that  $\pi_1$  is a valid proof for  $N_1$ , it generates a valid proof  $\pi$  for  $N_1 + N_2$ . To avoid trivialities, one requires that  $\pi$  is asymptotically succinct, even if the incrementation is applied arbitrarily many times, and that the running time of  $\text{Inc}$  is essentially independent<sup>2</sup> of  $N_1$ . This condition avoids the trivial solution, in which either  $\text{Inc}$  computes a proof  $\pi_2$  from  $\pi_1$  onwards

<sup>2</sup> By essentially independent we mean that the dependency is at most poly-logarithmic.

and defines  $\pi := (\pi_1, \pi_2)$ , or simply ignores  $\pi_1$  and computes  $\pi$  from scratch for  $N_1 + N_2$ .

The starting point for ITC is TC. Recall that TC is an interactive GL-PoSW whose underlying weighted DAG is  $(G_N = ([N]_0, E_G), \Omega_N)$ , where  $G_N$  is a tree-like DAG with a single sink  $N$  and  $\Omega_N$  is such that  $\Omega_N(i) := 1/N$  for every  $i \in [N]$ . The prover  $P$  computes, using a random oracle  $\tau(\cdot)$ , the labeling  $L$  of  $G_N$  and sends  $\phi_L := L(N)$  to  $V$ , which in return selects  $t$  challenges  $i_1, \dots, i_t$  according the distribution induced by  $\Omega_N$ , which in this case would be the uniform distribution, and sends them to  $P$ , which responds by openings to these challenges. Finally  $V$  accepts if and only if the openings are consistent with  $\phi_L$ , which serves as a commitment to  $L$ .

This construction is then made non-interactive using the Fiat-Shamir transform:  $P$  computes challenges by using enough random bits from applying the random oracle  $\tau(\chi, \phi_L, \cdot)$  appropriately many times, say on inputs  $1, 2, \dots$ , and then uses these random bits to deterministically generate  $t$  challenges. However, inherent to this approach, is that  $P$  only learns its challenges at the end of the computation, and by then, it either must have stored all  $L$  and can then open the challenges efficiently, or it recomputes the missing labels among  $L$  that are needed to answer the challenges. In between these two extremes there are general space-time tradeoffs that the prover can employ as we discussed above.

ITC is an alternative approach towards making TC non-interactive and incremental at the same time. ITC employs the clever *on-the-fly sampling* technique: as  $P$  is labeling  $G_N$ , in topological order, it learns some *potential* challenges from its already computed labels, and it learns with certainty what labels it already computed will not be part of its final challenges. This allows  $P$  to discard the labels of such useless nodes, and hence keep only the essential labels in memory. By the time  $P$  computes  $L(N)$ , it learns its final  $t$  challenges, for which it already stored their respective openings.

### 1.3 Incremental PoSWs for Incremental SNACKs

**On Light-Client Blockchain Protocols.** Recently, [2] show how GL-PoSW schemes can be generically used to augment blockchains such that they would then allow for efficient light-client secure bootstrapping. In their terminology, a full miner holding the entirety of a PoSW-augmented blockchain provides the light client with a *succinct non-interactive argument of chain knowledge* (SNACK) proof. The SNACK proof, in addition to some blockchain suffix blocks, allows the light client to hold a commitment to a stable prefix of the blockchain. For this application, a standalone GL-PoSW suffices: in the SNACK construction of [2], any GL-PoSW is used generically to augment a blockchain such that the augmented blockchain becomes SNACK-friendly, i.e., one can generate SNACK proofs for the augmented blockchain efficiently.

However, if the PoSW is incremental in the construction of [2], then the full node miner, having produced or obtained a *valid* SNACK proof for an augmented blockchain of length  $N$ , will be able to increment its SNACK proof for the

blockchain when it grows into any length  $N' > N$ , and hence allows the full node to bootstrap light clients without storing the first  $N$  blocks of the  $N'$ -long blockchain. (The genesis block is always assumed to be stored by all parties.) This allows full nodes to become space-efficient, which is a great advantage given the massive sizes of the ever-growing blockchains.

**The ITC Scheme and Light-Client Blockchain Protocols.** In [2], it is shown how to build a SNACK system from any GL-PoSW scheme. Furthermore, if the underlying GL-PoSW is incremental, then so is the overlying SNACK. The standalone (non-incremental) SNACK can be used to provide efficient solutions to the blockchain light-client bootstrapping problem. Additionally, if the SNACK is incremental, it is suggested in [2] that such an incremental SNACK would make the prover of these light-client protocols even more efficient; the prover becomes space-efficient.

SNACKs are defined with respect to a family of weighted DAG  $(G_N = ([N]_0, E_G), \Omega_N)_{N \geq 0}$ . The underlying weighted DAG of the SNACK is inherited from the underlying GL-PoSW. For the SNACK application of light-client bootstrapping,  $\Omega_N : [N] \rightarrow [0, 1]$  doesn't induce the uniform distribution over  $[N]$ . Roughly speaking,  $\Omega_N(i) \sim 1/(N - i)$ , which is far from uniform over  $[N]$ . This motivates designing incremental PoSW schemes with general weight distributions.

**On Defining Incremental PoSWs.** For the light-client blockchain application, an honest prover that increments *any valid SNACK proof* should make the verifier accept, regardless of how the SNACK proof it incremented was generated. This means that we need the same guarantees from the underlying iPoSW: *any valid PoSW proof* that is honestly incremented will make the verifier accept. This is also a natural requirement on incrementality, when the iPoSW is used in a distributed fashion: there are multiple parties that compute and increment; one party may increment another's computation and if the incrementation was done honestly on a valid proof, then the resulting proof must also verify regardless of how the proof of the previous party was generated.

## 1.4 Our Contributions

Motivated by the recent connection between light-client blockchain applications and GL-PoSW schemes [2], we advance the current understanding by

- strengthening the definition of iPoSW of [7] such that it becomes useful for light-client blockchain applications. The iPoSW definition of [7] only requires that honestly incrementing an honestly generated proof, rather than *any valid proof*, makes the verifier accept. For the usability of iPoSW in distributed applications, like incremental SNACKs and blockchains, we strengthen their definition as highlighted above. We also observe that their construction, ITC, achieves our stronger definition.
- constructing an iPoSW scheme whose underlying weighted graph family  $(G_N, \Omega_N)_{N \geq 0}$  is such that  $G_N$  is the simple skiplist  $G_N = ([N]_0, E_N)$  and

whose distribution  $\Omega_N$  is any arbitrary *t-incrementally sampleable* distribution. In particular, both the uniform and the SNACK distributions are *t-incrementally sampleable* distribution. Therefore, our iPoSW is the first and only iPoSW that can be used to construct an incremental SNACK, which in turn, can be used to construct the first *space-efficient* prover in blockchain light-client bootstrapping [2]. Along the way, we give a simple characterization of *t-incrementally sampleable* distributions. Technically, we also devise a new on-the-fly-sampling technique that works for all such distributions.

In a bit more detail. We first give a standalone iPoSW scheme based on the skiplist graph (Sect. 5). The scheme uses the same distribution  $\Omega_N$  as in [7] and can be thought of as a general-purpose iPoSW. To prove security, we use and adapt the same on-the-fly-sampling strategy of [7]. Informally speaking, and on a very high level, the on-the-fly sampling works by randomly and without replacement sampling a set  $S$  of  $t$  elements from two sets  $S_0$  and  $S_1$ , each contains  $t$  elements and a certain fraction of which is inconsistent. One then uses a Hoeffding bound to reason about the fraction of inconsistent elements in  $S$  in relation to the corresponding fractions of the original sets  $S_0, S_1$ . Fortunately, the same Hoeffding bound can be used when sampling randomly *with or without* replacement.

When moving to general probability distributions, one can still apply a general Hoeffding bound when sampling *with replacement* from a general probability distribution, i.e., when the random variables of the samples are independent. However, when sampling is done without replacement, then we are not aware of appropriate Hoeffding-like bounds that one can apply generically.

Therefore, we devise a new sampling strategy. Our sampling follows the Poisson binomial distribution, and informally, given  $S_0$  and  $S_1$  whose elements are sampled from  $\Omega_t$ , we sample  $S$  such that each  $s_i \in S_0 \cup S_1$  is added to  $S$  with probability  $p_i$  that is proportional to  $\Omega_{2t}(s_i)$ . Instead of having  $|S|$  exactly  $t$ , we have that  $|S|$  is on expectation  $t$ . We show that this sampling strategy works for all *t-incrementally sampleable* distributions.

In Sect. 6, we use this new on-the-fly sampling technique to construct an iPoSW which works for any *t-incrementally sampleable* distribution. When applying the new technique, new challenges arise: the verifier no longer can verify the consistency of the on-the-fly sampled elements. We solve this problem by making the prover commit to, and give away, as part of its proof, extra sets that allow the verifier to check the consistency of the sampled challenges. This change increases the proof size slightly.

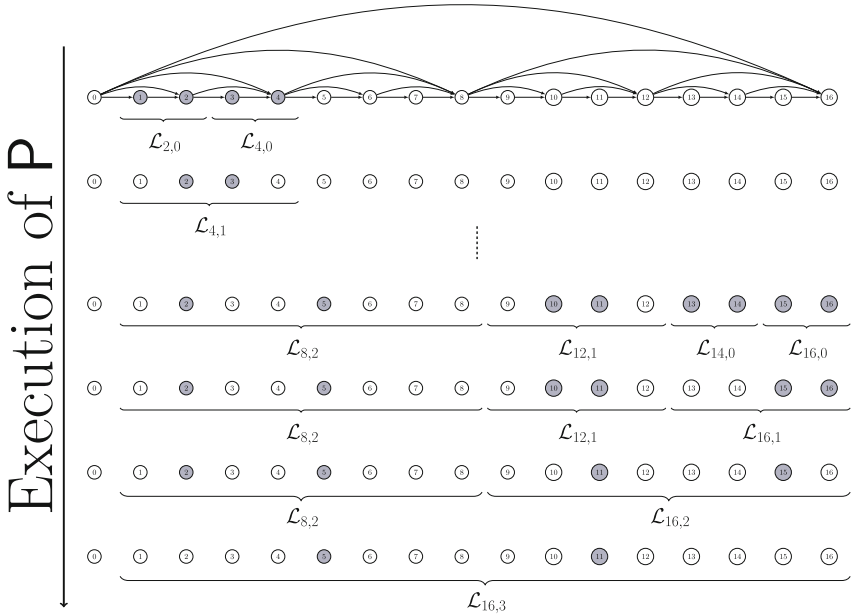
## 1.5 A High-Level Technical Overview

In this section, we only give the high-level overview of our skiplist-based iPoSW (Sect. 5) when  $\Omega_N$  is the uniform distribution and we sample without replacement. The general case when  $\Omega_N$  is any *t-incrementally sampleable* distribution and our on-the-fly sampling follows the Poisson binomial distribution is given in Sect. 6.2.



Let’s first review the interactive SC construction. Figure 1 shows the skiplist graph  $G_N = ([N], E_G)$  for  $N = 16$ . We only show how P, which has the labeling  $L$  of  $G_{16}$ , answers its challenges. For example, let  $v \in [16]$  be a challenge, then P locates the unique shortest path from the source 0 to the sink 16 which passes through  $v$ . Then it answers with the labels on this path and the labels of the parents of the nodes on the path. V simply checks the consistency of the labels on the path. As the path is of length  $O(\log N)$  and each node has  $O(\log N)$  parents, the opening for a single challenge is of size  $O(\lambda \log^2 N)$ . In contrast, the opening for a path in TC is  $O(\lambda \log N)$ .

Our incremental (non-interactive) PoSW ISC is then obtained by employing the on-the-fly sampling technique of [7]. For simplicity of exposition, we show how the challenges are on-the-fly sampled, however, without showing the corresponding openings of these challenges. In Sect. 5, we give a more comprehensive overview of the construction, including how to incrementally re/combine *partial* openings to *full* openings of the final challenges.



**Fig. 1.** Evolution of the stored challenges during the protocol. The elements of the list  $\mathcal{L}_{v,i}$  are actually full openings of challenges. To ease readability, we have only drawn the corresponding challenge nodes. An example for  $G_{16}$ ,  $t = 2$ .

The on-the-fly sampling is illustrated in Fig. 1 for an example graph  $G_{16}$  and  $t = 2$ . In general, the technique works as follow: let  $t = 2^c$  for some integer  $c$  be the number of challenge nodes/openings to be produced by the end of the protocol. For every  $v \in [N]$  that is a multiple of  $t$ , we will construct a list  $\mathcal{L}_{v,0}$  which contains all the nodes  $w \in [v - t + 1 : v]$ . This list consists of all

potential challenges defined by  $v$  at “level” 0. If  $v \in [N]$  is also a multiple of  $2t$ , the sets  $\mathcal{L}_{v-t,0}$  and  $\mathcal{L}_{v,0}$  are merged into a unique set, denoted by  $\mathcal{L}_{v,1}$ , with  $t$  elements, in the following way:  $L(v)$  is used as input to a random oracle to obtain random coins  $r_{v,1}$ . Using these random coins, we sample at random (and without replacement) a subset of size  $t$  from the set  $\mathcal{L}_{v-t,0} \cup \mathcal{L}_{v,0}$ . Such randomly sampled subset of size  $t$  will be stored as  $\mathcal{L}_{v,1}$ , and the sets  $\mathcal{L}_{v-t,0}$  and  $\mathcal{L}_{v,0}$  can be erased from memory. The set  $\mathcal{L}_{v,1}$  will consist of  $t$  challenge nodes assigned to  $v$  at “level” 2. In a similar way, whenever  $v$  is a multiple of  $2^i t$  for integer  $i$ , random coins  $r_{v,i}$  are produced from  $L(v)$  to obtain a random subset  $\mathcal{L}_{v,i}$  of size  $t$  from the set  $\mathcal{L}_{v-2^{i-1}t,i-1} \cup \mathcal{L}_{v,i-1}$ . After obtaining  $\mathcal{L}_{v,i}$ , the sets  $\mathcal{L}_{v-2^{i-1}t,i-1}$ , and  $\mathcal{L}_{v,i-1}$  are erased from memory. In the last step of the algorithm, the set  $\mathcal{L}_{N,n-c}$  will be produced, where  $N = 2^n$ . This, together with  $\phi_L := L(N)$ , constitutes the proof to be verified.

Extra care is needed when proving soundness of the resulting ISC scheme, in which the prover learns partial information about its challenges, even before sending the commitment  $\phi_L$ . The overall proof strategy is similar to proof of ITC from [7]. The intuition of why the on-the-fly sampling is sound comes from the observation that at each resampling node  $v$ , the samples are derived from randomness that depends on  $L(v)$ , and  $L(v)$  serves as a commitment to all potential nodes from which the resampling takes place. The security proof then goes through a series of hybrid games, in which two consecutive games differ by a resampling-related bad event, which happens with a negligible probability. The analysis of the last hybrid game boils down to the original analysis of the SC scheme.

As mentioned before, when moving to general weight distributions, our on-the-fly sampling procedure follows the Poisson binomial distribution. This change introduces a novel issue: the number of final challenges (and thus their total weight) is not fixed in advanced as before, and more critically, the intermediate sets used in the sampling procedures are not implicitly defined. On the one hand, it can be shown using Hoeffding-like bounds, that the number of challenges (and their total weight) must be concentrated around the expected value with overwhelming probability. By modifying the verifier so that it additionally checks such constraints, it is possible to rule out such malicious behavior, and fix the first issue. On the other hand, by having the prover commit to the intermediate sets, and give, as part of its proof, the opening of the intermediate sets used in sampling the final challenge set, the verifier can check the correctness of the on-the-fly sampled challenges, and thus fix the second issue as well. A more elaborate overview is given in Sect. 6.

## 2 Preliminaries

### 2.1 Notations

For integers  $m, n > 0$ , define  $[n] := \{1, \dots, n\}$ ,  $[n]_0 := [n] \cup \{0\}$ , and  $[m : n] := \{m, m + 1, \dots, n\}$ . For a DAG  $G = (V, E)$  and  $v \in V$ , we let  $\text{Parents}(v) := (v_1, \dots, v_k)$  be the parents of  $v$  given in *reverse topological ordering*.<sup>3</sup>

For a distribution  $\mathcal{D}$ , we denote by  $d \stackrel{\$}{\leftarrow} \mathcal{D}$  sampling  $d$  according to  $\mathcal{D}$  (in case  $\mathcal{D}$  is a set, the uniform distribution is implied).

We define the notion of (oracle-based) graph labeling which appeared in previous work on GL-PoSW schemes, say [5, 12].

### 2.2 Graph Labeling

**Definition 1 (Oracle-based graph labeling).** Let  $G_N = ([N]_0, E_N)$  be a DAG and  $\tau : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$  an oracle. We define the  $\tau$ -labeling  $L^\tau : [N]_0 \rightarrow \{0, 1\}^\lambda$  of  $G_N$  recursively as

$$L^\tau(i) := \begin{cases} \tau(i) & \text{if } \text{Parents}(i) = \emptyset \\ \tau(i, L^\tau(i_1), \dots, L^\tau(i_k)) & \text{else, i.e., } \text{Parents}(i) = (i_1, \dots, i_k) \end{cases} \quad (1)$$

When  $\tau$  is clear from the context, we simply write  $L$ .

To formalize consistency of labels in this context, [2] define the notion of consistent strings, which is stronger than prior definitions in the literature [3, 5, 12]. Intuitively, to each vertex  $i$  a value  $y_i$  is associated, which represents the concatenation of the labels of the *parents* of  $i$ . In order to reason about the label of the last node as well, a dummy vertex is introduced for it, that is, we add vertex  $N + 1$  and an edge  $(N, N + 1)$ .

**Definition 2 (Consistent strings [2]).** Let  $\tau : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$  be an oracle. For a DAG  $G_N = ([N]_0, E_N)$ , let  $G_N^+ = ([N + 1]_0, E_N^+)$  with  $E_N^+ = E_N \cup \{(N, N + 1)\}$ . Furthermore,  $\forall i \in [N + 1]_0$ , let  $p_i$  be the number of parents of  $i$  in  $G_N^+$  and  $y_i := (i, y_i[1], \dots, y_i[p_i]) \in ([N]_0 \times (\{0, 1\}^\lambda)^{p_i})$ . We say  $y_i$  is consistent with  $y_{i'}$  w.r.t.  $G_N$ , and denote it by  $y_i \prec y_{i'}$  if  $(i, i') \in E_N^+$  and if  $i$  is the  $j$ -th parent of  $i'$  in  $G_N^+$  (in reverse topological order), then the  $j$ -th block in the decomposition of  $y_{i'}$  is equal to  $\tau(y_i)$ , i.e.,  $y_{i'}[j] = \tau(y_i)$ .

## 3 The Skiplist PoSW Scheme

### 3.1 Construction

In this section, we review the interactive GL-PoSW scheme of [2]. The scheme is reminiscent to the PoSW scheme from [3]. The scheme follows the same design template introduced earlier and we give a formal definition only for the non-interactive incremental GL-PoSW schemes later in Sect. 4.

<sup>3</sup> Although any fixed ordering suffices, it would be convenient for our construction to consider the reverse topological ordering.

**Definition 3 (The Skiplist graph [2,3]).** Let  $G_N = ([N]_0, E_N)$  be a DAG with

$$E_n = \{(i, j) \in ([N]_0)^2 : \exists k \geq 0 \text{ s.t. } (j - i) = 2^k \wedge 2^k | i\} .$$

**Definition 4 (Labeled Paths).** Let  $G_N = ([N]_0, E_N)$  be a skiplist graph as in Def. 3 and  $L^\tau$  a labeling over its vertices for an oracle  $\tau : \{0, 1\} \rightarrow \{0, 1\}^\lambda$ . For integers  $i_1, \dots, i_j$  s.t.  $0 \leq i_1 < \dots < i_j \leq N$ , define  $\text{Path}(i_1, \dots, i_j)$  as the unique shortest path from  $i_1$  to  $i_j$  passing through  $i_1, \dots, i_j$ . Furthermore, define

$$\begin{aligned} \text{Path}^+(i_1, \dots, i_j) &:= (\text{Parents}(v))_{v \in \text{Path}(i_1, \dots, i_j)} \\ \text{Path}^*(i_1, \dots, i_j) &:= (v, L(\text{Parents}(v)))_{v \in \text{Path}(i_1, \dots, i_j)} \end{aligned}$$

and a predicate **Consistent** over labeled paths as follow:

**Consistent** ( $\text{Path}^*(i_1, \dots, i_j) \in \{0, 1\}$ ): output 1 iff

$\forall y_i := (v_i, \cdot), y_{i'} := (v_{i'}, \cdot) \in \text{Path}^*(i_1, \dots, i_j)$  s.t.  $(v_i, v_{i'}) \in E_N$ :

$$y_i \prec y_{i'} \quad \text{where } \prec \text{ is as in Def. 2.}$$

To illustrate, consider  $G_8$ , then

$$\begin{aligned} \text{Path}(0, 3, 8) &= (0, 2, 3, 4, 8) \\ \text{Path}^+(0, 3, 8) &= ((), (1, 0), (2), (3, 2, 0), (7, 6, 4, 0)) \\ \text{Path}^*(0, 3, 8) &= (((0), (2, L(1), L(0)), (3, L(2)), (4, L(3), L(2), L(0)), \\ &\quad (8, L(7), L(6), L(4), L(0)))) . \end{aligned}$$

Furthermore,  $\text{Consistent}(\text{Path}^*(0, 3, 8)) = 1$  as

$$y_0 \prec y_2, y_0 \prec y_4, y_0 \prec y_8, y_2 \prec y_3, y_2 \prec y_4, y_3 \prec y_4, y_4 \prec y_8.$$

Note that  $\text{Path}^*$  contains redundant labels. This results in an increase in the proof size in the skiplist-based PoSW scheme. However, we keep it as is for the simplicity of exposition. In practical realizations, it is straightforward to remove the redundancy and modify the verification algorithm accordingly.

Now we describe the PoSW scheme from [2] in Fig. 2. Formally the PoSW is defined for a graph family  $(\Gamma_N := (G_N, \Omega_N))_{N \in \mathbb{N}}$  where  $G_N$  is a skiplist as in Def. 3 and a weight function  $\Omega_N : [N] \rightarrow [0, 1]$  where  $\sum_{i \in [N]} \Omega_N(i) = 1$ . Besides, the PoSW scheme is parameterized by a security parameter  $t \in \mathbb{N}$  that determines the number of challenges the verifier samples from  $\Omega_N$ .

In [2], interactive GL-PoSW schemes were defined. Their definition generalizes existing definitions of PoSW in a few directions. First, the protocol's underlying graph is a weighted DAG  $\Gamma_n := (G_N, \Omega_N)$  where  $G_N$  is a DAG on  $[N]_0$  vertices, and the the verifier's challenges are drawn according to a weigh function  $\Omega_N : [N]_0 \rightarrow [0, 1]$  where  $\sum_{i \in [N]_0} \Omega_N(i) = 1$ . Second, they define *knowledge* soundness in addition to the classical notion of soundness. In Theorem 1, we

<p><u>Prover P = (P<sub>0</sub>, P<sub>1</sub>) :</u></p> <p><b>Stage P<sub>0</sub>:</b> On input <math>(1^\lambda, 1^N)</math> and <math>\chi</math>:</p> <ol style="list-style-type: none"> <li>1. Compute the <math>\tau</math>-labeling <math>L</math> of <math>G_N</math> using oracle <math>\tau(\chi, \cdot)</math></li> <li>2. <math>\phi_L := L(N)</math></li> <li>3. <b>send</b> <math>\phi_L</math> to <math>V_1</math></li> </ol> <p><b>Stage P<sub>1</sub>:</b></p> <ol style="list-style-type: none"> <li>1. <math>\forall i \in [t], y_i := \text{Path}^*(0, \iota_i, N)</math> where <math>\text{Path}^*</math> is as in Def. 4.</li> <li>2. <b>send</b> <math>y := (y_1, \dots, y_t)</math> to <math>V_2</math></li> </ol>	<p><u>Verifier V = (V<sub>0</sub>, V<sub>1</sub>, V<sub>2</sub>):</u></p> <p><b>Stage V<sub>0</sub>:</b> On input <math>(1^\lambda, N)</math>:</p> <ol style="list-style-type: none"> <li>1. <math>\chi \leftarrow \{0, 1\}^\lambda</math></li> <li>2. <b>send</b> <math>\chi</math> to <math>P_0</math></li> </ol> <p><b>Stage V<sub>1</sub>:</b> On input <math>\phi_L</math>:</p> <ol style="list-style-type: none"> <li>1. <math>\forall i \in [t]</math> <b>do</b> <math>\iota_i \xleftarrow{\\$} \Omega_N</math></li> <li>2. <b>send</b> <math>\iota = (\iota_i)_{i=1}^t</math> to <math>P_1</math></li> </ol> <p><b>Stage V<sub>2</sub>:</b> On input <math>y = (y_i)_{i=1}^t</math>:</p> <ol style="list-style-type: none"> <li>1. <math>\forall i \in [t]</math> <b>do</b> <ol style="list-style-type: none"> <li>(a) parse <math>y_i = (y_{i_1}, \dots, y_{i_k})</math> where <math>(i_1, \dots, i_k) := \text{Path}(0, \iota_i, N)</math></li> <li>(b) <math>b_i := 1</math> iff                     <ol style="list-style-type: none"> <li>i. <math>\tau(y_{i_k}) = \phi_L</math> and</li> <li>ii. <math>\text{Consistent}(y_i) = 1</math></li> </ol>                     where <math>\text{Consistent}</math> is as in Def. 4.                 </li> </ol> </li> <li>2. <b>output</b> <math>\bigwedge_{i=1}^t b_i</math></li> </ol>
---	--

**Fig. 2.**  $t \in \mathbb{N}$  is a parameter of the scheme, and  $(G_N, \Omega_N)$  is s.t.  $G_N$  is a skiplist DAG as in Def. 3 and  $\Omega_N : [N] \rightarrow [0, 1]$  is a weight function, i.e., it satisfies  $\sum_{i \in [N]} \Omega_N(i) = 1$ .

restate their main theorem about the classical, rather than knowledge, soundness guarantees of the PoSW construction depicted in Fig. 2.

Towards generalizing PoSW to arbitrary weight functions, the weight of a sequence of parallel oracle queries to  $\tau(\cdot)$  is defined. A parallel query is a sequence of simultaneous queries to  $\tau$ , i.e. a sequence  $((x_1, i_1), \dots, (x_m, i_m))$  which is answered by  $(\tau(i_1, x_1), \dots, \tau(i_m, x_m))$ . Intuitively, the weight of such a sequence is the sum of the respective “heaviest” nodes in each parallel query.

**Definition 5 (Sequential weight [2]).** Let  $Q = (Q_1, \dots, Q_\ell)$  be a sequence of parallel queries to an oracle  $\tau$ . We define the sequential weight of  $Q$  with respect to a weight function  $\Omega_N : [N] \rightarrow [0, 1]$  where  $\sum_{i \in [N]} \Omega_N(i) = 1$  as

$$\Omega_{seq}(Q) := \sum_{i=1}^{\ell} \max\{\Omega_N(j) : Q_i \text{ contains a query to } \tau(j, \cdot)\} .$$

The honest P in Fig. 2 defines a sequence  $Q = (Q_1, \dots, Q_N)$  of queries with  $Q_i := \{(i, L(\text{Parents}(i)))\}$  and it therefore holds that  $\Omega_{seq}(Q) = 1$ . To capture the sequential work of malicious provers, the notion of  $\tau$ -sequence is defined.

Sequentiality of random oracles is formulated in terms of RO-sequences, which appeared in slightly different formulations and bounds in [2, 3, 5, 12]. Below we adopt the formulation from [2].

**Definition 6 ( $\tau$ -sequences [2]).** Let  $G_N = ([N]_0, E_N)$  be a DAG and  $\tau : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$  be a random oracle. We call a sequence of strings  $s :=$

$(y_{i_1}, \dots, y_{i_{\ell+1}})$  with  $y_{i_{\ell+1}} \in \{0, 1\}^\lambda$  and  $y_{i_j} \in ([N]_0 \times (\{0, 1\}^\lambda)^{|\text{Parents}(i_j)|})$  a  $\tau$ -sequence of length  $\ell$  if  $\forall j \in [\ell], y_{i_j} \prec y_{i_{j+1}}$  w.r.t.  $G_N$ . ( $\prec$  is as in Def. 2.)

For a weight function  $\Omega_N : [N] \rightarrow [0, 1]$ , the weight of a  $\tau$ -sequence  $s = (y_{i_1}, \dots, y_{i_{\ell+1}})$  is defined as  $\Omega_N(s) := \sum_{j=1}^{\ell} \Omega(i_j)$ .

Note the honest  $Q$  above can be made into a  $\tau$ -sequence  $s := (y_0, \dots, y_N, y_{N+1})$  where  $\forall i \in [N]_0, y_i := (i, L(\text{Parents}(i)))$  and  $y_{N+1} := \tau(y_N)$ . Note the weight of  $s$  is defined to ignore the last index  $N+1$  and hence is equal  $\sum_{i=1}^N \Omega(i) = 1$ . Furthermore, note if  $\text{Path}^*(0, i, N)$  is such that  $\text{Consistent}(\text{Path}^*(0, i, N)) = 1$ , then by definition  $(\text{Path}^*(0, i, N), \phi_L)$  constitutes a  $\tau$ -sequence. Lemma 1 shows that, except with negligible probability, no malicious prover which makes a sequence of parallel queries  $Q$  with  $\Omega_{\text{seq}}(Q) < \alpha$  can produce a  $\tau$ -sequence of weight  $\alpha$ .

**Lemma 1. (Sequentiality of  $\tau$  [2]).** *Let  $\Gamma_N = (G_N, \Omega_N)$  be a weighted DAG and  $\tau : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$  a random oracle. Let  $\tilde{P}^{\tau(\cdot)}$  be a malicious prover that makes (parallel) queries to  $\tau(\cdot)$  of sequential weight  $< \alpha$  and makes  $q$  oracle queries in total. Then the probability that  $\tilde{P}^{\tau(\cdot)}$  outputs a  $\tau$ -sequence of weight  $\alpha$  can be bounded by*

$$\Pr \left[ s \leftarrow \tilde{P}^{\tau(\cdot)} : s \text{ is a } \tau\text{-sequence} \wedge \Omega_N(s) = \alpha \right] \leq \frac{1}{2^\lambda} + \frac{q^2}{2^\lambda} = \frac{q^2 + 1}{2^\lambda}.$$

Now we state the theorem that shows the PoSW scheme in Fig. 2 is secure.

**Theorem 1. ([2]).** *Consider a malicious prover  $\tilde{P}$  against  $\mathbb{V}$  from Fig. 2. If  $\tilde{P}$  makes a sequence  $Q$  of parallel queries to  $\tau$  of sequential weight  $\Omega_{\text{seq}}(Q) < \alpha \in (0, 1]$  and a total number of queries  $q$ , then  $\tilde{P}$  can make  $\mathbb{V}$  accept with probability at most  $\epsilon := \alpha^t + 3 \cdot q^2 / 2^\lambda$ .*

### 3.2 Prover Efficiency and Space-Time Tradeoffs

In this section, we give general space-time tradeoffs on the prover’s strategy in the interactive PoSW from [2]. These tradeoffs pave the way to the incremental non-interactive PoSW scheme we give in Sect. 5. For space constraints, we give the proofs of the lemmas of this section in the full version of the paper.

The following lemma shows that the prover can label the skiplist graph in small space complexity.

**Lemma 2.** *Let  $G_N$  be the skiplist graph from Def. 3 and  $\tau : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$  an oracle. Then  $G_N$  for  $N = 2^n$  can be  $\tau$ -labeled in topological order using at most  $(n + 1)\lambda$  bits of memory*

Let  $N = 2^n$  and the level of a node denote its in-degree. In  $G_N$ , we have only the source node 0 with level 0, only the sink node  $N$  with level  $n + 1$ , and for every  $i \in [n]$ , we have  $2^{n-i}$  nodes with level  $i$ . For simplicity of exposition, we treat the source node differently and assume that its label is always known. When storing labels of nodes of level  $m$  and greater, the prover needs to store the labels of  $\sum_{i=m}^n 2^{n-i} + 1 = 2^{n-m+1}$  nodes. Recomputing the labels of the

remaining nodes can be done by making  $2^{m-1} - 1$  queries sequentially; this can be done assuming  $2^{n-m+1}$  parallel processors by partitioning  $2^n$  into intervals of length  $2^{m-1}$ . To compute how many (sequential) queries are required to compute all the labels necessary to correctly reply to some challenge  $i \in [N]$ , we need to first analyze how many nodes' labels will be needed in answering that challenge.

**Lemma 3.** *Let  $G_N$  be the graph from Def. 3 with  $N = 2^n$ , and let  $k \in [n + 1]$ . If a challenge hits a  $k$ -level node, then the prover can convince the verifier by providing the labels of  $\phi_n(k) := 2 + \frac{1}{2}(n - k + 2)(n + k - 1)$  vertices.*

The following lemma says that, when storing labels of all nodes of level greater than or equal to  $m$ , given  $\rho(n, m, k)$  parallel processors, it is possible to reply to the challenge query making only  $2^{m-1} - 1$  sequential hash queries.

**Lemma 4.** *If a challenge hits a  $k$ -level node,  $k \in [n + 1]$ , of a graph with  $N = 2^n + 1$  nodes, where  $n \geq 2$ , then a prover storing the labels of all nodes of level greater than or equal to  $m$ , for  $m \in [n+1]$ , will have to make  $\rho(n, m, k) \cdot (2^{m-1} - 1)$  many queries, where*

$$\rho(n, m, k) := \begin{cases} 0 & \text{if } m = 0, \\ n - k + 2 & \text{if } m < k, \\ n - m + 2 & \text{otherwise.} \end{cases}$$

### 4 Incremental Proofs of Sequential Work

We provide a definition of incremental proofs of sequential work that is stronger than the definition given by [7]. In fact, their construction as well as ours achieve the stronger definition. Our definition is stronger in the sense that it guarantees that honestly incrementing *any valid proof*, regardless of how it is generated, makes the verifier accept.

This issue is particularly important in the context of graph-labeling proofs of sequential work schemes as these are not proofs of *correctness* of the sequential computation, but rather are proofs of sequential computation. Consider any graph-labeling PoSW with underlying graph  $G_N$  and a prover that follows the honest prover strategy except for a randomly chosen  $i \leftarrow [N]$ , it sets  $L(i) = 0^\lambda$  and continues the computation correctly. Such a prover has an overwhelming probability of convincing the verifier, although the proof was not honestly generated, and with overwhelming probability the proof will be different from the honestly generated proof. However, this is not a problem of the concept of a PoSW, as still the malicious prover did  $N - 1$  computational steps sequentially.

**Definition 7 (Incremental PoSW).** *Let  $\Gamma = (\Gamma_N = (G_N, \Omega_N))_{N \in \mathbb{N}}$  be a family of weighted DAGs such that for all  $N$ ,  $G_N$  has a unique sink  $N$ . A tuple of oracle aided PPT algorithms  $(P^{\tau(\cdot)}, \text{Inc}^{\tau(\cdot)}, V^{\tau(\cdot)} := (V_0^{\tau(\cdot)}, V_1^{\tau(\cdot)}))$  for an oracle  $\tau : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$  is an incremental (non-interactive) proof of sequential work w.r.t.  $\tau$  if the following properties hold:*

**Completeness:** For every  $\lambda, N \in \mathbb{N}$ , every  $(\chi, N, \text{state}) \leftarrow V_0^{\tau(\cdot)}(1^\lambda, N)$  and

- honestly generated  $\pi \leftarrow P^{\tau(\cdot)}(1^\lambda, 1^N, \chi)$  or
- honestly incremented  $\pi \leftarrow \text{Inc}^{\tau(\cdot)}(1^\lambda, 1^{N''}, \chi, N', \pi')$  for integers  $N', N''$  s.t.  $N = N' + N''$  and  $\pi'$  is accepting<sup>4</sup> proof for parameter  $N'$ , i.e.,  $V_1^{\tau(\cdot)}(\text{state}, \chi, N', \pi') = 1$ ,

it holds that  $\Pr \left[ V_1^{\tau(\cdot)}(\text{state}, \chi, N, \pi) = 1 \right] \geq 1 - \text{negl}(\lambda)$ .

**$(\alpha, \epsilon)$ -Soundness:** For every  $\lambda, N \in \mathbb{N}$  and every PPT adversary  $\tilde{P}^{\tau(\cdot)}$  which makes a sequence  $Q$  of parallel queries to  $\tau$  of sequential weight  $\Omega_{\text{seq}}(Q) < \alpha$ :

$$\Pr \left[ \begin{array}{l} (\chi, N, \text{state}) \leftarrow V_0^{\tau(\cdot)}(1^\lambda, N); \\ \pi \leftarrow \tilde{P}^{\tau(\cdot)}(1^\lambda, 1^N, \chi) \end{array} : V_1^{\tau(\cdot)}(\text{state}, \chi, N, \pi) = 1 \right] \leq \epsilon(\lambda).$$

**Succinctness:** For every  $\lambda, N \in \mathbb{N}$  and every honestly generated proof  $\pi$  for parameter  $N$ , we have  $|\pi| \leq \text{poly}(\lambda, \log N)$  and  $V^{\tau(\cdot)}$  runs in time  $\text{poly}(\lambda, \log N)$ .

*Remark 1.* For  $\text{Inc}^{\tau(\cdot)}$  to be non-trivial, note that  $\text{Inc}^{\tau(\cdot)}$  runs in time that is essentially independent of  $N'$ ; it gets  $N'$  in binary.

*Remark 2.* For some applications, standard  $(\alpha, \epsilon)$ -soundness might not be enough, and *knowledge* soundness might be required. It is straightforward to define knowledge soundness for Def. 7 exactly the same way knowledge soundness of GL-PoSW from [2] is defined.

## 5 A Skiplist-Based Incremental PoSW Scheme

In this section, we construct an incremental GL-PoSW scheme based on the skiplist graph  $G_N, \Omega_N$  where  $\Omega_N$  is defined as  $\forall i \in [N] : \Omega_N(i) = 1/N$ . A construction of an iPoSW for more general families of weight distributions  $\Omega_N$  is given in Sect. 6.

In Sect. 5.2, we will give a high-level overview of our construction, using some concrete example to better explain our design. A formal description of the algorithms can be found in Sect. 5.3. The security proof is in Sect. 5.5.

### 5.1 Parameters

Our iPoSW scheme depends on the following parameters and objects.

- A time parameter  $N$  of the form  $N = 2^n$ , for some integer  $n \in \mathbb{N}$ .
- A computational security parameter  $\lambda$ .
- A statistical security parameter  $t$ .

---

<sup>4</sup> We consider  $\text{Inc}^{\tau(\cdot)}$  to be honest regardless of how  $\pi'$  was generated. In contrast, [7] defines honest  $\text{Inc}^{\tau(\cdot)}$  only over honestly generated  $\pi'$ .



- Let  $\tau : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$  be a random oracle, we define two oracles,  $\tau_\ell, \tau_r$  as
 
$$\tau_\ell(\cdot) := \tau(0, \chi, \cdot) \quad \tau_r(\cdot) := \tau(1, \chi, \cdot) \tag{2}$$
- $(i_1, \dots, i_t) := \text{Sample}(r)$ : on input random coins  $r$ , uniformly sample  $(i_1, \dots, i_t)$  from all possible such sequences  $\binom{2t}{t}$ . Since  $\binom{2t}{t} < (\frac{2t-e}{t})^t = (2e)^t$ , where  $\log_2(2e) \approx 2.44$ , random coins of size  $3t$  are sufficient to sample statistically close to a uniform subset.

*Remark 3.* Suppose that we have two sets  $S_1, S_2$  with  $t$  elements each, where in  $S_1$  the elements are indexed from 1 to  $t$ , and in  $S_2$  from  $t + 1$  to  $2t$ . Moreover, suppose that there are  $M_1$  elements having a *specific feature* in  $S_1$  and  $M_2$  having the same feature in  $S_2$ . Then, using **Sample**, one can sample a subset from  $S_1 \cup S_2$  of size  $t$ , where the distribution of the cardinality of elements with this same specific feature is described by a random variable  $X$  distributed as **Hypergeometric**( $2t, M_1 + M_2, t$ ).

For simplicity of exposition, we assume that  $t = 2^c$  for some  $c \in \mathbb{N}$ . Our construction can be easily adapted to the more general case where  $t$  is arbitrary.

## 5.2 A High-Level Overview

Before formally describing the algorithms defining the iPoSW scheme, we give an intuition of how they work.

We start by describing the PoSW scheme upon which we build. The scheme is described in Fig. 2. In such interactive protocol, the first step is performed by the verifier  $V$ , which samples a random statement  $\chi$  with enough min entropy and sends it to the prover  $P$ , which in turn uses  $\chi$  to refresh the common random oracle  $\tau(\cdot)$  as  $\tau_\ell(\cdot) := \tau(0, \chi, \cdot)$  and computes a  $\tau_\ell$ -labeling  $L$  of  $G_N$ . Then  $P$  sends  $\phi_L := L(N)$  to  $V$ . As shown in [2],  $\phi_L$  constitutes a (position-binding) commitment to  $L$ .<sup>5</sup> After receiving  $\phi_L$ ,  $V$  samples  $t$  challenge nodes and sends them to  $P$ , which in turn sends to  $V$  valid openings for all the received challenges. In the last step of the interaction,  $V$  checks the consistency of the openings and accepts or rejects accordingly.

This protocol can be made non-interactive using the Fiat-Shamir heuristic: the challenges chosen by  $V$  are now produced by  $P$  itself by querying a random oracle  $\tau_r(\cdot) := \tau(\chi, 1, \cdot)$  on  $(\phi_L, 1), \dots, (\phi_L, t)$ . Now  $V$ , besides verifying the consistency of the openings, must also verify the correctness of the received challenges, by recomputing them based on  $\phi_L$  and  $\tau_r(\cdot)$ .

Using this protocol, in order to compute a proof, the prover has to either remember the  $N$  labels for the entire  $G_N$  graph, or recompute the labels required in the proof, once the challenge nodes are fixed, which may require up to  $N$  sequential invocations to  $\tau_\ell$ . Alternatively, it is possible to have general space-time trade-offs, as shown in Lemma 4: the prover upon spending  $N$  sequential

---

<sup>5</sup> This is reminiscent to the fact that a Merkle tree’s root commits to its leaves (and internal nodes).

invocation to  $\tau_r$ , could store  $\approx 2^{n-m}$  labels from  $L$ , and perform an additional  $\approx 2^{m-1}$  sequential computation using  $\rho(n, m, k) \leq n - m + 2$  parallel processors, for any  $m \in [n + 1]$ . For example, taking  $m = n/2$ , yields a protocol where the prover uses  $\approx \sqrt{N}$  memory, and  $\approx N + \sqrt{N}/2$  sequential work, while using  $\approx (\log N)/2$  processors.

Still, there is a concrete substantial gap between, on the one hand, the sequential work the honest prover has to actually perform and, on the other hand, the sequential work it is able to convince the verifier of. For example, a memory-efficient prover, i.e., one where  $m = n$ , has to perform  $2N$  sequential queries to  $\tau_r$  in order to show it did  $N$  sequential queries. For large values of  $N$ , this gap becomes considerable; say  $N$  steps would take approximately a year to compute, then to generate a proof of that, one would need another year.

As first observed by Döttling, Lai and Malavolta in [7], at the core of this slack is the fact that the challenge nodes are determined solely by  $\phi_L$ . Therefore, the prover has to label the entire graph before knowing which labels it has to recompute and include in the proof.

The problem was first tackled by [7], who introduced the idea of letting the prover choose the challenge nodes *on the fly*: the prover chooses the random challenges as it labels the graph, and eventually discards some of them as the graph gets labeled. This allows the prover to compute a valid proof using *a single pass* over the graph. However, since now the prover knows partial information about the possible challenges while labeling the graph, soundness of the so obtained protocol has to be carefully studied.

We adapt the on-the-fly sampling technique of [7] to the skiplist-based PoSW scheme. Let  $t$  be the number of challenge nodes/openings to be produced by the end of the protocol (for ease of notation we will consider  $t = 2^c$  to be a power of 2). For each node  $v$  in  $[N]$  which is a multiple of  $t$ , we will construct a list  $\mathcal{L}_{v,0}$  which contains all the nodes  $w \in [v - t + 1, v]$ . This list will represent the challenge node assigned to  $v$  at “level” 0.

When  $v$  is also a multiple of  $2t$ , the sets  $\mathcal{L}_{v-t,0}$  and  $\mathcal{L}_{v,0}$  are merged into a unique set, denoted by  $\mathcal{L}_{v,1}$ , with  $t$  elements, in the following way:  $L(v)$  is used as input to the random oracle  $\tau_r$  to obtain random coins  $r_{v,1}$ . Using these random coins, we sample a subset  $\mathcal{L}_{v,1}$  of size  $t$  uniformly at random from  $\mathcal{L}_{v-t,0} \cup \mathcal{L}_{v,0}$ . Once  $\mathcal{L}_{v,1}$  is stored,  $\mathcal{L}_{v-t,0}$  and  $\mathcal{L}_{v,0}$  are erased from memory. The set  $\mathcal{L}_{v,1}$  consists of  $t$  challenge nodes assigned to  $v$  at “level” 2. In a similar way, whenever  $v$  is a multiple of  $2^i t$ , random coins  $r_{v,i}$  are produced from the labeling of  $v$ , to obtain a random subset  $\mathcal{L}_{v,i}$  of size  $t$  from the set  $\mathcal{L}_{v-2^{i-1}t, i-1} \cup \mathcal{L}_{v, i-1}$ . After obtaining  $\mathcal{L}_{v,i}$ , the sets  $\mathcal{L}_{v-2^{i-1}t, i-1}$ , and  $\mathcal{L}_{v, i-1}$  are erased from memory.

In the last step of the algorithm, the set  $\mathcal{L}_{N, n-c}$  will be produced. This, together with  $\phi_L = L(N)$ , constitute the proof to be verified.

We depict the sampling process pictorially for graph  $G_{16}$  and parameter  $t = 2$ , i.e.,  $n = 4$ , and  $c = 1$ , in Fig. 1. The prover algorithm P labels all nodes of the graph in topological order. Once it has labeled the first 2 nodes, it creates the set  $\mathcal{L}_{2,0}$ , which contains the nodes 1 and 2 as possible challenges. It then continues to label the graph. When it reaches node 4, it first creates the set

$\mathcal{L}_{4,0}$ , which contains the nodes 3 and 4 as possible challenges. From the union of these two sets, a random subset of 2 elements is chosen to obtain the set  $\mathcal{L}_{4,1}$ . In the last step of the protocol, lists  $\mathcal{L}_{16,0}$ ,  $\mathcal{L}_{14,0}$ ,  $\mathcal{L}_{12,1}$ , and  $\mathcal{L}_{8,2}$  are merged in succession to finally obtain  $\mathcal{L}_{16,3}$ .

As already anticipated, in the actual protocol the lists  $\mathcal{L}_{w,j}$ , for nodes  $w \in [N]$  and  $j \in [n-c]$  contain more than simple challenge nodes. Each element in  $\mathcal{L}_{w,j}$  is an opening (a labeled path) for a challenge node in  $[w - 2^j t + 1 : w]$  with respect to  $G_N|_{[w-2^j t:w]}$ , where  $G|_{V'}$ , with  $V' \subset V$ , denotes the subgraph of  $G$  induced by the vertex set  $V'$ , i.e., the graph  $G'$  with vertex set  $V'$  and edge set consisting of those edges both of whose endpoints are in  $V'$ .

Given  $\mathcal{L}_{v-2^{i-1}t, i-1}$  and  $\mathcal{L}_{v, i-1}$  with openings (labeled paths), rather than challenges, we show how to construct  $\mathcal{L}_{v, i}$ . As before, using  $L(v)$  we extract randomness to sample a random subset of size  $t$  from  $\mathcal{L}_{v-2^{i-1}t, i-1} \cup \mathcal{L}_{v, i-1}$ . Moreover, suppose that the first element sampled in this way is  $\mathcal{L}_{v, i-1}[b]$ , for some  $b \in [t]$ . Now  $\mathcal{L}_{v, i-1}[b]$  is an opening of some challenge node in  $[v - 2^{i-1}t + 1 : v]$  with respect to  $G_N|_{[v-2^{i-1}t:v]}$ . Since elements in  $\mathcal{L}_{v, i}$  have to be valid openings of some challenge node in  $[v - 2^i t + 1 : v]$  with respect to  $G_N|_{[v-2^i t:v]}$ , simply adding  $\mathcal{L}_{v, i-1}[b]$  to  $\mathcal{L}_{v, i}$  won't work. Instead, we extend  $\mathcal{L}_{v, i-1}[b]$  to a valid opening over the whole subgraph  $G_N|_{[v-2^i t:v]}$ . This can be done, using the structure of the skiplist graph and the definition of shortest path, by simply pre-appending the missing labeled edge  $(v - 2^{i-1}t, L(\text{Parents}(v - 2^{i-1}t)))$  to  $\mathcal{L}_{v, i-1}[b]$ . Similarly, any element in  $\mathcal{L}_{v-2^{i-1}t, i-1}$  can be extended to a valid opening in  $G_N|_{[v-2^i t:v]}$  by appending  $(v, L(\text{Parents}(v)))$  to it. This step of the algorithm is formally described by (4) in the scheme description.

The prover's final proof is  $\pi = (\phi_L, \mathcal{L}_{N, n-c}, \mathcal{I}_{N, n-c})$ . For each  $b \in [t]$ , the verifier checks that (1)  $\mathcal{L}_{N, n-c}[b]$  is consistent with  $\phi_L$  (2) that  $\mathcal{L}_{N, n-c}[b]$  is internally consistent, and (3) that the challenge in  $\mathcal{L}_{N, n-c}[b]$  is consistent with the on-the-fly-sampling. The first two checks are similar to the base PoSW scheme. To check (3), algorithm **Check** of Fig. 3 is run: on input  $\mathcal{L}_{v, i}[b]$  and  $\mathcal{I}_{v, i}[b]$ , it recomputes the randomness  $r_{v, i}$  used to sample elements from  $\mathcal{L}_{v-2^{i-1}t, i-1}[b]$  and  $\mathcal{L}_{v, i-1}[b]$ , checks if this is consistent with what is stored in  $\mathcal{I}_{v, i}[b]$ , and determines whether  $\mathcal{L}_{v, i}[b]$  was selected from  $\mathcal{L}_{v-2^{i-1}t, i-1}[b]$  or from  $\mathcal{L}_{v, i-1}[b]$ . If  $\mathcal{L}_{v, i}[b]$  was obtained by extending some element of  $\mathcal{L}_{v-2^{i-1}t, i-1}[b]$  to  $G_N|_{[v-2^i t, v]}$ , then it must be the case that the last two nodes are exactly  $v - 2^{i-1}t$  and  $v$ . If this is not the case, the algorithm returns immediately 0. Otherwise,  $(v, L(\text{Parents}(v)))$ , gets removed from it, and **Check** is run recursively on the so obtained opening.

### 5.3 Scheme Description

$\mathbf{P}^{\tau(\cdot)}(1^\lambda, 1^N, N)$ :

1. Traverse the graph  $G_n = (V = [N]_0, E)$  in topological order, starting from 0. At every node  $v \in [N]_0$  which is traversed, do the following:
  - (a) Compute  $L(v)$  according to (1).
  - (b) If  $t \mid v$  and  $v \in [N]$ , write  $v = 2^k \cdot h \cdot t$ , with  $h$  odd and  $k \in \mathbb{N}_0$ . For  $j \in [t]$ :

$$\mathcal{L}_{v,0}[j] := \text{Path}^*(v-t, v-t+j, v), \quad \mathcal{I}_{v,0}[j] := j. \quad (3)$$

If  $k \geq 1$ , do the following for  $i \in [k]$ :

- i. Compute  $r_{v,i} := \tau_r(v, i, L(v))$ .
- ii. Choose a random  $t$ -subset  $S_{v,i}$  of  $[2t]$  via  $S_{v,i} := \text{Sample}(r_{v,i})$ .
- iii. Set  $u := v - 2^{i-1} \cdot t$ .
- iv. For  $j \in [t]$ , write  $S_{v,i}[j] = at + b$  with  $a \in \{0, 1\}$  and  $b \in [t]$ , and set

$$\mathcal{L}_{v,i}[j] := \begin{cases} (\mathcal{L}_{u,i-1}[b], (v, L(\text{Parents}(v)))) & \text{if } a = 0 \\ ((u, L(\text{Parents}(u))), \mathcal{L}_{v,i-1}[b]) & \text{if } a = 1 \end{cases} \quad (4)$$

$$\mathcal{I}_{v,i}[j] := \begin{cases} \mathcal{I}_{u,i-1}[b], j & \text{if } a = 0 \\ \mathcal{I}_{v,i-1}[b], j & \text{if } a = 1 \end{cases}$$

- v. Store  $\mathcal{L}_{v,i}$  and  $\mathcal{I}_{v,i}$  in memory. Note that by design,  $\mathcal{L}_{v,i}[j]$  satisfies

$$\text{Consistent}(\mathcal{L}_{v,i}[j]) = 1. \quad (5)$$

- vi. Erase  $\mathcal{L}_{u,i-1}$ ,  $\mathcal{L}_{v,i-1}$ ,  $\mathcal{I}_{u,i-1}$ , and  $\mathcal{I}_{v,i-1}$  from memory.

2. Terminate and output  $\pi := (\phi_L := L(N), \mathcal{L}_{N,n-c}, \mathcal{I}_{N,n-c})$ .

$\text{Inc}^{\tau(\cdot)}(1^\lambda, 1^{N'}, N, \pi, \chi)$ :

1. Parse  $\pi$  as  $(\phi_L, \mathcal{L}_{N,n-c}, \mathcal{I}_{N,n-c})$ .
2. Compute  $N + N' = N''$  and check that  $N'' = 2^{n''}$  for some  $n'' \in \mathbb{N}$ .
3. Execute the algorithm  $\text{P}^{\tau(\cdot)}(1^\lambda, 1^{N''}, \chi)$  starting from step 1.(a) with a slight change: traverse the graph  $G_{2^{n''}}$  starting from  $N + 1$ .

$\text{V}^{\tau(\cdot)}(1^\lambda, 1^N, \pi, \chi)$ :

1. Parse  $\pi$  as  $(\phi_L, \mathcal{L}_{N,n-c}, \mathcal{I}_{N,n-c})$ .
2. For all  $i \in [t]$ :
  - (a) parse  $\mathcal{L}_{N,n-c}[i]$  as  $y_i = (y_{i_1}, \dots, y_{i_k})$ , for some  $k \in \mathbb{N}$ .
  - (b)  $b_i := 1$  iff the following hold
    - i.  $\tau_\ell(y_{i_k}) = \phi_L$  and
    - ii.  $\text{Consistent}(y_i) = 1$  where  $\text{Consistent}$  is as in Def. 4.
    - iii.  $\text{Check}(\mathcal{L}_{N,n-c}[i], \mathcal{I}_{N,n-c}[i], 0, N, n - c) = 1$ , where the algorithm  $\text{GCheck}$  is described in Fig. 3.
3. Return  $\bigwedge_{i=1}^t b_i$

Algorithm Check

On input a path  $\text{Path}$ , a list of indices  $\text{ind}$ , a starting node  $\mathbf{s}$ , an ending node  $\mathbf{e}$ , and a recursion index  $\mathbf{d}$ :

1. Parse  $\text{Path} := (y_1, \dots, y_k)$  and  $\text{ind} := (i_0, \dots, i_d) \in [t]^{d+1}$
2. Compute  $r_{\mathbf{e},\mathbf{d}} := \tau_r(\mathbf{e}, \mathbf{d}, \tau_\ell(y_k))$  and  $S_{\mathbf{e},\mathbf{d}} := \text{Sample}(r_{\mathbf{e},\mathbf{d}})$
3. Write  $S_{\mathbf{e},\mathbf{d}}[i_d] = at + b$  with  $a \in \{0, 1\}$  and  $b \in [t]$
4. Define  $\mathbf{m} := (\mathbf{e} - \mathbf{s})/2$
5. If  $\mathbf{d} = 0$ , do the following:
  - Compute the challenge  $c$  corresponding to  $\text{Path}$
  - If  $a = 0$ , return 1 iff  $\mathbf{s} + b = c$
  - If  $a = 1$ , return 1 iff  $\mathbf{s} + \mathbf{m} + b = c$
6. If  $\mathbf{d} \geq 1$ , do the following
  - If  $a = 0$  and  $i_{d-1} = b$ ,  $\text{Check}((y_1, \dots, y_{k-1}), (i_1, \dots, i_{d-1}), \mathbf{s}, \mathbf{s} + \mathbf{m}, \mathbf{d} - 1)$
  - If  $a = 1$  and  $i_{d-1} = b$ ,  $\text{Check}((y_2, \dots, y_k), (i_1, \dots, i_{d-1}), \mathbf{s} + \mathbf{m}, \mathbf{e}, \mathbf{d} - 1)$
  - Return 0

**Fig. 3.** Description of the Check algorithm.

### 5.4 Efficiency Analysis

We now discuss the efficiency of our scheme in terms of proof size, computation, and communication.

*Proof Size.* The proof consists of the sink-label  $\phi_L$  and two lists,  $\mathcal{L}_{N,n-c}$  and  $\mathcal{I}_{N,n-c}$ , with  $t$  elements each. Each entry in  $\mathcal{L}_{N,n-c}$  is a path of the form  $\text{Path}^*(0, i, N)$  for some  $i \in [N]$ . By Lemma 3, it therefore consists of at most  $2 + \frac{n(n+1)}{2} = O(n^2)$  labels and  $n + 1 = O(n)$  indices. Each entry in  $\mathcal{I}_{N,n-c}$  is a tuple of  $n - c$  indices in  $[t]$ , which can therefore be represented using  $\log t$  bits each. Since each label can be stored using  $\lambda$  bits, we get that the entire proof has size at most  $O(t \cdot (\lambda \cdot n^2 + n)) = O(t \cdot \lambda \cdot n^2)$ .

*Prover Efficiency.* The prover traverses the  $N$  nodes of the skiplist graph  $G_N$  in topological order. By the same argument used in [7], the challenges  $\tau_r(\cdot)$  can be evaluated in parallel by computing  $r_{v,i} := \tau_r(v, i, L(\text{Parents}(v)))$ , instead of  $r_{v,i} := \tau_r(v, i, L(v))$ . This is possible as both  $\tau_\ell$  and  $\tau_r$  are random oracles. With such modification,  $\tau_\ell$  and  $\tau_r$  can be evaluated in parallel, thus the parallel complexity is not increased by the evaluation of  $\tau_r$ . Therefore, the parallel complexity of the prover is bounded by the time needed for  $O(N)$  sequential calls to the random oracle.

As far as the memory complexity of the prover is concerned, by Lemma 2 we know that the labeling of the skiplist graph  $G_N$  can be computed using  $(n + 1)\lambda$  bits of memory. In our construction, the prover algorithm is moreover storing at most  $n + 1$  lists  $\mathcal{L}_{v,i}$ , where at most 2 of them at any point share the same “level”  $i$ . Using Lemma 3 to compute the memory required to store all such lists, one obtains that the memory complexity of the prover is bounded by  $O(t \cdot \lambda \cdot n^3)$ .

*Verifier Efficiency.* The verifier need to check, for each opening of a challenge node that i) the opening is consistent with  $\phi_L$ , ii) that the labels of the opening are consistent, and that iii) the opening is consistent with the randomness used while generating the proof. Each opening can be checked in parallel. Checking consistency of each label in a given opening can also be done in parallel. The runtime of the verifier is dominated by the runtime of the Check algorithm, which uses  $O(n)$  time to verify that the opening is consistent with the randomness used in the proof. Therefore, the parallel time needed overall is  $O(n \log t)$ .

## 5.5 The Security Proof

**Theorem 2.** *Let  $\Pi := (\mathsf{P}^{\tau(\cdot)}, \mathsf{V}^{\tau(\cdot)}, \text{Inc}^{\tau(\cdot)})$  be as in Sect. 5.3 and  $q$  be an upper bound on the total number of queries to  $\tau(\cdot)$ , then  $\Pi$  is an  $(\alpha, \epsilon)$ -soundness incremental PoSW scheme, as per Def. 7, with any  $\alpha \in (0, 1]$  and*

$$\epsilon = \frac{1 + q^2}{2^\lambda} + \frac{q(q-1)}{2^{\lambda+1}} + q \cdot e^{-2t \cdot (\frac{1-\alpha}{n})^2} .$$

We remark that the weighted skiplist DAG  $(G_N, \Omega_N)$  over which  $\Pi$  works is such that  $\Omega_N$  is defined as  $\forall i \in [N] : \Omega_N(i) = 1/N$ . This induces the uniform distribution over the  $[N]$  challenges.

*Proof.* (of Theorem 2) Completeness and succinctness are clear from the discussion in Sect. 5.4. We analyze soundness. The soundness guarantees of our construction rely on the soundness guarantees of the original interactive PoSW construction from Sect. 5.3 modulo a number of hybrids that reflect the more power the adversary  $\tilde{\mathsf{P}}$  has in the security experiment of the incremental PoSW scheme. This power comes from the fact that  $\tilde{\mathsf{P}}$  gets to know some of its possible challenges early on during its computation. More precisely, before computing the label of the sink node which defines the set of challenges,  $\tilde{\mathsf{P}}$  gets to know that some nodes would not belong to its challenges. We will show that this extra power gives  $\tilde{\mathsf{P}}$  only a negligible advantage over the interactive counterpart PoSW. We start by describing the hybrid games, the last of which, almost corresponds to the security experiment in the interactive PoSW scheme.

$\text{Exp}_{\tilde{\mathsf{P}}, 0}^{\tau(\cdot)}(1^\lambda, 1^N) \in \{0, 1\}$ : Sample  $\chi \leftarrow \{0, 1\}^\lambda$ , run  $\pi \leftarrow \tilde{\mathsf{P}}^{\tau(\cdot)}(1^\lambda, 1^N, \chi)$ , and observe its queries  $Q_{\tau_\ell} \cup Q_{\tau_r}$  where  $Q_{\tau_\ell} := \{(x, y) : y = \tau_\ell(x)\}$  and  $Q_{\tau_r} := \{(x, y) : y = \tau_r(x)\}$  and  $\tau_\ell, \tau_r$  are as in (2). Use  $Q_{\tau_\ell}$  to build the query graph  $\text{QG} := (V, E)$ .

The query graph has as vertices the queries in  $Q_{\tau_\ell}$  and an edge is added between two vertices if and only if these two vertices have a corresponding edge in the skiplist graph  $G_N := ([N]_0, E_N)$  and that the queries are consistent on that edge. Formally, let  $V, E := \emptyset$ , then we populate them as follows:

$$\text{For every } v_i := (x_i, y_i), v_j := (x_j, y_j) \in Q_{\tau_\ell},$$

add  $v_i, v_j$  to  $V$  and the edge  $(v_i, v_j)$  to  $E$  iff  $x_i \prec x_j$  w.r.t. the skiplist graph  $G_N$  and the operator  $\prec$  as in Def. 2. If  $(x_j = (N, x'_j), y_j) \in V$  for some  $x'_j$ , then

check whether  $y_j = \phi_L$  and if not, remove  $v_j$  from  $V$  and all its incoming edges; note that  $y_j = \phi_L$  implies that  $x_j \prec \phi_L$ .

If  $\pi$  is invalid output 0, otherwise let  $\pi := (\phi_L, \mathcal{L}_{N,n-c}, \mathcal{I}_{N,n-c})$  be a valid proof:  $\forall i \in [t]$ , parse  $\mathcal{L}_{N,n-c}[i]$  as  $z_i = (z_{i_1}, \dots, z_{i_k})$  for some  $k \in \mathbb{N}$ . As  $\pi$  is a valid proof, it holds that  $\text{Consistent}(z_i) = 1$  and that  $\tau_\ell(z_{i_k}) = \phi_L$ . By definition of **Consistent**, this means that

$$s_i := (z_{i_1}, \dots, z_{i_k}, \phi_L) , \tag{6}$$

forms a  $\tau_\ell$ -sequence according to Def. 6.

Finally define the output of the experiment to be 1 if and only if  $\pi$  is valid and  $\forall i \in [t]$ ,  $s_i$  is *extractable* from **QG**. Formally, we say that  $s_i := (z_{i_1}, \dots, z_{i_k}, \phi_L)$  is extractable from **QG** if

$$\forall j \in [k-1], (z_{i_j}, y_{i_j}) \in V \text{ and that } (z_{i_k}, \phi_L) \in V . \tag{7}$$

$\text{Exp}_{\tilde{P},1}^{\tau(\cdot)}(1^\lambda, 1^N) \in \{0, 1\}$ : This experiment is identical to  $\text{Exp}_{\tilde{P},0}^{\tau(\cdot)}(1^\lambda, 1^N)$  except that  $\text{Exp}_{\tilde{P},1}^{\tau(\cdot)}(1^\lambda, 1^N)$  doesn't check the winning condition above that requires that  $\forall i \in [t]$ ,  $s_i$  is extractable from **QG**, i.e.,  $\text{Exp}_{\tilde{P},1}^{\tau(\cdot)}(1^\lambda, 1^N) = 1$  iff  $\pi$  is valid.

**Proposition 1.** *Let  $q$  upper-bounds the total number of queries  $\tilde{P}^{\tau(\cdot)}$  makes to  $\tau : \{0, 1\} \rightarrow \{0, 1\}^\lambda$ , then*

$$\left| \Pr \left[ \text{Exp}_{\tilde{P},0}^{\tau(\cdot)}(1^\lambda, 1^N) = 1 \right] - \Pr \left[ \text{Exp}_{\tilde{P},1}^{\tau(\cdot)}(1^\lambda, 1^N) = 1 \right] \right| \leq \frac{1}{2^\lambda} + \frac{q(q-1)}{2^{2\lambda+1}} . \tag{8}$$

The proof of Proposition 1 boils down to either finding a collision under  $\tau_\ell$  from at most  $q$  queries, or that  $\tilde{P}$  can guess the output of  $\tau_\ell$ . The latter happens with probability  $1/2^\lambda$  and the former with probability  $q(q-1)/2^{2\lambda+1}$ . The formal proof is given in the full version of the paper.

Recall that  $t = 2^c$  for some integer  $c$  and  $N = 2^n$  and that  $t \mid N$ . We define the challenge sampling set  $\mathcal{D}$  and the challenge re-sampling set  $\mathcal{C}$  as follow. These two sets will define a series of games that the security proof will go through.

$$\begin{aligned} \mathcal{D} &:= \{(v, 0) : \forall v \in [N] \text{ s.t. } t \mid v\} \\ \mathcal{C} &:= \{(v, 1), \dots, (v, k) : \forall v \in [N] \text{ s.t. } v = 2^k \cdot h \cdot t \text{ for odd } h \text{ and } k \geq 1\} \end{aligned}$$

Examples of such (ordered) sets would be

$$\begin{aligned} \mathcal{D} &= \{(t, 0), (2t, 0), (3t, 0), (4t, 0), (5t, 0), \dots\} \\ \mathcal{C} &= \{(2t, 1), (4t, 1), (4t, 2), (6t, 1), \dots, (N, 1), \dots, (N, n-c-1), (N, n-c)\} \end{aligned}$$

For  $\beta := (v, i) \in \mathcal{D} \cup \mathcal{C}$ , the prover algorithm from Sect. 5.3 implicitly defines a set of associated  $t$  challenges, call it  $\text{chal}(\beta)$ , and computes for each such set, the corresponding set of labeled paths, denoted as  $\mathcal{L}_\beta$ . This is formally described in (3) and (4), however, for readability's sake, we elaborate upon it below. For

$\beta := (v, 0) \in \mathcal{C}$ , the prover algorithm defines  $\text{chal}(\beta) := \{v, v - 1, \dots, v - t + 1\}$  and no resampling is needed. However, for  $\beta := (v, i) \in \mathcal{C}$ , i.e.,  $i \geq 1$ ,  $\text{chal}(\beta)$  is resampled from  $\text{chal}(\beta_0)$  and  $\text{chal}(\beta_1)$  where by construction<sup>6</sup>  $\beta_0 := (v, i - 1)$  and  $\beta_1 := (u, i - 1)$ . The resampling is according to the hypergeometric distribution as done by **Sample**.

For  $\beta \in \mathcal{D} \cup \mathcal{C}$ , we define functions  $\delta, \gamma, \eta : \mathcal{D} \cup \mathcal{C} \rightarrow [0, 1]$ , event  $\text{bad}_\beta$ , security experiment  $\text{Exp}_{\tilde{P}, \beta}^{\tau(\cdot)}$ , and  $\text{neighbor} : \mathcal{D} \cup \mathcal{C} \rightarrow \mathcal{D} \cup \mathcal{C}$ .

- $\gamma(\beta)$ : the fraction of inconsistent nodes among all possible nodes that could have been included into  $\text{chal}(\beta)$ . (Inconsistent in the sense that if  $v_i \in \text{chal}(\beta)$  and its corresponding path in  $\mathcal{L}_\beta$  is  $\text{Path}_{v_i}^*$ , then  $\text{Consistent}(\text{Path}_{v_i}^*) = 0$ .) For example, for  $\beta = (N, n - c)$ , it holds that  $\gamma(\beta)$  equals the number of all inconsistent nodes among  $[N]$  divided by  $N$ .
- $\delta(\beta)$ : the fraction of inconsistent nodes in  $\text{chal}(\beta)$ . We will be interested in analyzing how close  $\delta(\beta)$  is to  $\gamma(\beta)$ .
- For  $\alpha$  from Theorem 2, define

$$\eta(\beta) := \frac{i}{n - c} \cdot (1 - \alpha) . \tag{9}$$

- Event  $\text{bad}_\beta$  is defined whenever a re/sampling takes place, i.e., when  $\tau_r(\beta, \cdot)$  is called

$$\text{bad}_\beta := 1 \Leftrightarrow \delta(\beta) < \gamma(\beta) - \eta(\beta) . \tag{10}$$

- $\text{neighbor}(\beta) := \beta'$ : Let  $\mathcal{C}^* \subseteq \mathcal{C}$  be the (ordered) set on whose elements  $\tilde{P}$  issued resampling queries, i.e.,  $\tau_r(\beta, \cdot)$  is a resampling query on  $\beta \in \mathcal{C}^*$ . If  $\beta$  is the first such element in  $\mathcal{C}^*$ , then set  $\beta' = 1$ , and otherwise  $\beta'$  is defined to be the previous element in  $\mathcal{C}^*$ .
- $\text{Exp}_{\tilde{P}, \beta}^{\tau(\cdot)}(1^\lambda, 1^N) \in \{0, 1\}$ : this is identical to its neighboring  $\text{Exp}_{\tilde{P}, \beta'}^{\tau(\cdot)}(1^\lambda, 1^N)$  except it outputs 0 if  $\text{bad}_\beta = 1$ .

**Proposition 2.** *For every  $\beta \in \mathcal{D} \cup \mathcal{C}$  and  $\beta' := \text{neighbor}(\beta)$ , the following holds*

$$\left| \Pr \left[ \text{Exp}_{\tilde{P}, \beta}^{\tau(\cdot)}(1^\lambda, 1^N) = 1 \right] - \Pr \left[ \text{Exp}_{\tilde{P}, \beta'}^{\tau(\cdot)}(1^\lambda, 1^N) = 1 \right] \right| \leq e^{-2t \cdot \left(\frac{1-\alpha}{n-c}\right)^2} . \tag{11}$$

Before proving Proposition 2, we make a few observations. For the first  $\beta \in \mathcal{C}^*$ , it holds that  $\text{Exp}_{\tilde{P}, \beta'}^{\tau(\cdot)} = \text{Exp}_{\tilde{P}, 1}^{\tau(\cdot)}$ , and the last experiment corresponds to  $\beta = (N, n - c) \in \mathcal{C}$ . As the total number of such experiments is at most  $q$ , and the distance between each neighboring experiments is bounded by Proposition 2, it then holds by a simple union bound that

$$\left| \Pr \left[ \text{Exp}_{\tilde{P}, 1}^{\tau(\cdot)}(1^\lambda, 1^N) = 1 \right] - \Pr \left[ \text{Exp}_{\tilde{P}, (N, n-c)}^{\tau(\cdot)}(1^\lambda, 1^N) = 1 \right] \right| \leq q \cdot e^{-2t \cdot \left(\frac{1-\alpha}{n-c}\right)^2} . \tag{12}$$

Observe that for the final game with  $\beta = (N, n - c)$  corresponds to the sink vertex  $G_N$ . If  $\text{bad}_\beta = 0$ , then the soundness analysis is similar to the analysis

<sup>6</sup> See Sect. 5.3 and/or Fig. 1.



of the interactive PoSW given in [2], and recalled in Sect. 5.3. More concretely, by definition it follows that  $\eta(\beta) = (1 - \alpha)$ . Now if  $\tilde{P}$  made  $Q_{\tau_\ell}$  queries of sequential weight  $\Omega_{seq}(Q_{\tau_\ell}) < \alpha$ , then by Lemma 1, except with probability  $\epsilon_{\tau_\ell} := 1/2^\lambda + q^2/2^\lambda$ , this must mean that  $\gamma(\beta) > (1 - \alpha)$ , which then implies that  $\delta(\beta) > 0$ , which implies, that  $V$  rejects the proof, as the proof will contain at least one inconsistent path. Therefore,

$$\Pr \left[ \text{Exp}_{\tilde{P}, (N, n-c)}^{\tau(\cdot)}(1^\lambda, 1^N) = 1 \right] \leq \epsilon_{\tau_\ell} := \frac{1}{2^\lambda} + \frac{q^2}{2^\lambda} . \tag{13}$$

Before bounding the final probability, we observe that in the probability  $\epsilon_{\tau_\ell}$  above,  $1/2^\lambda$  accounts for guessing the output of  $\tau_\ell$ . And as we accounted for such event in analyzing Proposition 1, we conclude that the probability  $\epsilon$  in Theorem 2 can be upper bounded by summing the probability in (8), (11), (12), (13) and subtracting  $1/2^\lambda$ . This concludes the proof.  $\square$

**Proof of Proposition 2.** The proof makes use of the following lemma, which is simply a restatement from [7].

**Lemma 5.** *Let  $t$  be an integer,  $b \in \{0, 1\}$ ,  $\delta_b \in [0, 1]$ , and sets  $U_b$  s.t.  $|U_b| = t$ ,  $U_0 \cap U_1 = \emptyset$  and  $U_b$  contains elements that are either consistent or inconsistent with  $\delta_b$  being the fraction of inconsistent elements in  $U_b$ , and*

$$\delta_b \geq \gamma_b - \eta_b , \tag{14}$$

for some global values  $\gamma_b, \eta_b \in [0, 1]$ .

We sample from  $U_0 \cup U_1$  without replacement in  $t$  draws a set  $U$ , with  $|U| = t$ , and let  $X$  be the random variable indicating the number of inconsistent elements in  $U$ . Then an arbitrary  $\eta \in [0, 1]$ , we have

$$\Pr \left[ X \leq t \cdot \left( \frac{\gamma_0 + \gamma_1}{2} - \eta \right) \right] \leq e^{-2t \cdot (\eta - \frac{\gamma_0 + \gamma_1}{2})^2} . \tag{15}$$

The proof of Lemma 5 uses a Hoeffding bound and is given in the full version of the paper.

*Proof. (of Proposition 2).* The proof amounts to bounding the probability of  $\text{bad}_\beta$ . We have two cases: If  $\beta = (v, 0) \in \mathcal{D}$ , then by definition,  $\text{bad}_\beta = 0$  as  $\eta(\beta) = 0$  and  $\delta(\beta) = \gamma(\beta)$ . If  $\beta = (v, i) \in \mathcal{C}$ , then we resample  $t$  elements according to  $\text{Sample from } \text{chal}(\beta_0) \cup \text{chal}(\beta_1)$  for some  $\beta_0 = (v, i - 1)$  and  $\beta_1 = (u, i - 1)$  for which  $\text{bad}_{\beta_0} = \text{bad}_{\beta_1} = 0$ , for otherwise we would not be in this game. Then we have that  $\gamma_b := \gamma(\beta_b), \delta_b := \delta(\beta_b), \eta_b := \eta(\beta_b)$  satisfy

$$\delta_b \geq \gamma_b - \eta_b = \gamma_b - \frac{i - 1}{n - c} \cdot (1 - \alpha) . \tag{16}$$

Now applying Lemma 5 on  $U_b := \text{chal}(\beta_b)$ ,  $\eta := \eta(\beta)$  and noting that  $\gamma := \gamma(\beta) := (\gamma_0 + \gamma_1)/2$ , we get

$$\Pr[\text{bad}_\beta = 1] = \Pr[t \cdot \delta(\beta) < t \cdot (\gamma - \eta)] \leq e^{-2t \cdot \left( \frac{i \cdot (1-\alpha)}{n-c} - \frac{(i-1) \cdot (1-\alpha)}{n-c} \right)^2} = e^{-2t \cdot \left( \frac{1-\alpha}{n-c} \right)^2} .$$

$\square$

## 6 Incremental PoSW for General Distributions

In this section, we give our skiplist-based iPoSW for general weight distributions. We characterize these distributions as  $t$ -incrementally sampleable distributions. The construction is similar to the construction of Sect. 5, except that, we devise and use a new on-the-fly sampling technique, which samples according to the Poisson Binomial distribution. The need for the new sampling technique is motivated by applying Hoeffding-like tail bounds when sampling from general distributions, rather than sampling without replacement uniformly at random as is the case for the construction from Sect. 5 and that of [7]. The new sampling technique introduces a small modification to the main construction: the prover now needs to additionally give out, as part of its proof, subsets that are used for the resampling algorithm. These are needed for the verifier to validate the consistency of the resampling. Moreover, the verifier will have to check that number of nodes (and their weight) of such subsets is within appropriate intervals centered around their expectation.

*Our on-the-Fly Sampling Technique.* Let  $\Omega := \{\Omega_{2^i \cdot t} : [2^i \cdot t] \rightarrow [0, 1]\}_{i \geq 0}$  be a family of (weight) distributions,  $U_0 \subseteq [2^i \cdot t]$  and  $U_1 \subseteq [2^i \cdot t + 1 : 2^{i+1} \cdot t]$  sets sampled according to  $t \cdot \Omega_{2^i \cdot t}$ . (Technically, as the domains of  $\Omega_{2^i \cdot t}$  and  $U_1$  mismatch, we think of the  $2^i \cdot t$ -shifted  $U_1$  as being sampled from  $t \cdot \Omega_{2^i \cdot t}$ .)

The goal of the on-the-fly sampling is to sample  $W$  from  $U := U_0 \cup U_1$  such that  $W$  is distributed according to  $t \cdot \Omega_{2^{i+1} \cdot t}$ . That is, instead of directly sampling  $W$  from  $[2^{i+1} \cdot t]$  according to  $t \cdot \Omega_{2^{i+1} \cdot t}$ , the on-the-fly sampling allows one to first sample each  $U_0$  and  $U_1$  individually according to  $t \cdot \Omega_{2^i \cdot t}$ , and then sample  $W$  from  $U$ . For the sampling of  $W$  from  $U$  to be possible, it necessary that every  $u \in U$  is at least as probable in  $\Omega_{2^i \cdot t}$  as in  $\Omega_{2^{i+1} \cdot t}$ . This is reflected in conditions 2 and 3 in Def. 8. Furthermore, technically, it must be that  $t \cdot \Omega_{2^i \cdot t} \leq 1$  for every  $i$ . This is implied by conditions 1, 2 and 3 in Def. 8.

The  $t$  factor in the sampling process above is stipulated by the security of iPoSW schemes, which requires the prover to open  $t$  challenges. This corresponds to requiring that  $W$  has  $t$  samples. Our technique ensures this on expectation.

GSample (Fig. 4) formalizes how  $W$  is sampled from  $U$ . Note in Fig. 4,  $W$  consists of indices of elements from  $U$ , rather than the actual elements.

### 6.1 Incrementally Sampleable Distributions

We characterize weight distributions that can be on-the-fly sampled. These are  $t$ -incrementally sampleable (weight) distributions.

**Definition 8.** For  $t \in \mathbb{N}^+$  and a family of weight functions  $\Omega := \{\Omega_{2^i \cdot t} : [2^i \cdot t] \rightarrow [0, 1]\}_{i \geq 0}$ , we say  $\Omega$  is  $t$ -incrementally sampleable if the following hold:

1.  $\forall j \in [t] : t \cdot \Omega_t(j) \leq 1$
2.  $\forall i \geq 0, \forall j \in [2^i \cdot t] : \Omega_{2^{i+1} \cdot t}(j) \leq \Omega_{2^i \cdot t}(j)$
3.  $\forall i \geq 0, \forall j \in [2^i \cdot t + 1 : 2^{i+1} \cdot t] : \Omega_{2^{i+1} \cdot t}(j) \leq \Omega_{2^i \cdot t}(j - 2^i \cdot t)$

It is immediate to see that the uniform distribution on  $[N]$  is  $t$ -incrementally sampleable for any  $t \leq N$ . Another distribution that is of particular interest to the application of our incremental PoSW to incremental SNACKs is the SNACK distribution of [2]. For a fixed positive integer  $\ell$  and every positive integer  $m$ , the SNACK distribution  $\Omega_m : [m] \rightarrow [0, 1]$  is defined as

$$\Omega_m(j) = S_m \cdot \frac{1}{m + \ell - j} \quad \text{where} \quad S_m := \left( \sum_{j=1}^m \frac{1}{m + \ell - j} \right)^{-1}. \quad (17)$$

**Lemma 6.** *The SNACK distribution  $\Omega := \{\Omega_{2^{i \cdot t}}\}_{i \geq 0}$  where  $\Omega_{2^{i \cdot t}}$  is as in (17) is  $t$ -incrementally sampleable for  $\ell \geq t \cdot S_t$ .*

The proof is elementary and is given in the full version of the paper.

### 6.2 Scheme Description

Our iPoSW scheme for  $t$ -sampleable distributions is in spirit similar to the iPoSW scheme from Sect. 5. The main difference is that we employ our new on-the-fly sampling technique. This has correctness and security consequences that we address.

Recall that the on-the-fly sampling technique from Sect. 5 always produces  $W$  of size exactly  $t$  from intermediate sets  $U_0$  and  $U_1$  which are implicitly defined. This allows the verifier  $V$  to check the correctness of  $W$ . In contrast, our on-the-fly sampling produces  $W$  whose size is  $t$  on expectation, and more critically, the intermediate sets  $U_0$  and  $U_1$  are not implicitly defined, and hence, the prover  $P$  has to give  $U_0$  and  $U_1$  for  $V$  to verify the correctness of  $W$ . This results in an increase in the proof size: for each challenge  $v$ ,  $P$  gives  $\log N$  pairs of sets with total expected size  $2 \cdot t \cdot \log^2 N$ . The increased proof size is still succinct though.

This could potentially allow a malicious prover  $\tilde{P}$  to manipulate the sets  $U_0$  and  $U_1$  such that the challenges in  $W$  don't include challenges in  $U_0 \cup U_1$  that the adversary can't correctly answer. To get around this issue,  $P$  commits to the sampling sets across the execution of the protocol. This ensures that the sets in different challenges are fixed and consistent with each other. However, this doesn't rule out that  $\tilde{P}$  would not gain any advantage by committing to manipulated sets. We address this issue in the security proof by showing that except with a negligible probability,  $\tilde{P}$  gains no advantage in cheating on the sampling sets.

We give the formal construction with similar parameters as in Sect. 5.1 and the following additional parameters:  $t_{\min} := (1 - \zeta) \cdot t$  and  $t_{\max} := (1 + \zeta) \cdot t$  for  $\zeta \in (0, 1)$ ,  $\omega_{\min, d, \Omega} \in (0, 1)$  and  $\omega_{\max, d, \Omega} \in (0, 1)$  for  $d \in \mathbb{N}$ .

$$\underline{P^{\tau(\cdot)}(1^\lambda, 1^N, N, \Omega)}:$$

1. Traverse the graph  $G_n = (V = [N]_0, E)$  in topological order, starting from 0. At every node  $v \in [N]_0$  which is traversed, do the following:
  - (a) Compute  $L(v)$  according to (1).

(b) If  $t \mid v$  and  $v \in [N]$ , write  $v = 2^k \cdot h \cdot t$ , with  $h$  odd and  $k \in \mathbb{N}_0$ . For  $j \in [t]$ :

$$\begin{aligned} \mathcal{L}_{v,0}[j] &:= \text{Path}^*(v-t, v-t+j, v), & \mathcal{I}_{v,0}[j] &:= j \\ U_{v,0}[j] &:= v-t+j, & \mathcal{U}_{v,0}[j] &:= \perp \end{aligned} \quad (18)$$

If  $k \geq 1$ , do the following for  $i \in [k]$ :

- i. Compute  $u := v - 2^{i-1} \cdot t$
- ii. Compute  $h_0 := |U_{u,i-1}|$ ,  $h_1 := |U_{v,i-1}|$
- iii. Compute  $r_{v,i} := \tau_r(v, i, L(v), U_{u,i-1}, U_{v,i-1})$ .
- iv. Choose a subset  $S_{v,i}$  from  $[h_0 + h_1]$  as  
 $S_{v,i} := \text{GSample}(i, \Omega, t, U_{u,i-1}, U_{v,i-1}; r_{v,i})$  with  $\text{GSample}$  in Fig. 4.
- v. For  $j \in [|S_{v,i}|]$ , do the following

$$a := \begin{cases} 0 & \text{if } S_{v,i}[j] \leq h_0 \\ 1 & \text{if } S_{v,i}[j] > h_0 \end{cases} \quad \text{and} \quad b := S_{v,i}[j] - h_0 \cdot a$$

$$\mathcal{L}_{v,i}[j] := \begin{cases} (\mathcal{L}_{u,i-1}[b], (v, L(\text{Parents}(v)))) & \text{if } a = 0 \\ ((u, L(\text{Parents}(u))), \mathcal{L}_{v,i-1}[b]) & \text{if } a = 1 \end{cases} \quad (19)$$

$$\mathcal{I}_{v,i}[j] := \begin{cases} \mathcal{I}_{u,i-1}[b], j & \text{if } a = 0 \\ \mathcal{I}_{v,i-1}[b], j & \text{if } a = 1 \end{cases}$$

$$\mathcal{U}_{v,i}[j] := \begin{cases} \mathcal{U}_{u,i-1}[b], U_{u,i-1}, U_{v,i-1} & \text{if } a = 0 \\ U_{v,i-1}[b], U_{u,i-1}, U_{v,i-1} & \text{if } a = 1 \end{cases}$$

$$U_{v,i}[j] := \begin{cases} U_{u,i-1}[b] & \text{if } a = 0 \\ v, i-1[b] & \text{if } a = 1 \end{cases}$$

- vi. Store  $\mathcal{L}_{v,i}$ ,  $\mathcal{I}_{v,i}$ ,  $\mathcal{U}_{v,i}$ , and  $U_{v,i}$  in memory. Note that by design,  $\mathcal{L}_{v,i}[j]$  satisfies  $\text{Consistent}(\mathcal{L}_{v,i}[j]) = 1$ .
  - vii. For  $x \in \{u, v\}$ , erase from memory  $\mathcal{L}_{x,i-1}$ ,  $\mathcal{I}_{x,i-1}$ ,  $\mathcal{U}_{x,i-1}$ , and  $U_{x,i-1}$ .
2. Terminate and output  $\pi := (\phi_L := L(N), \mathcal{L}_{N,n-c}, \mathcal{I}_{N,n-c}, \mathcal{U}_{N,n-c}, U_{N,n-c})$ .

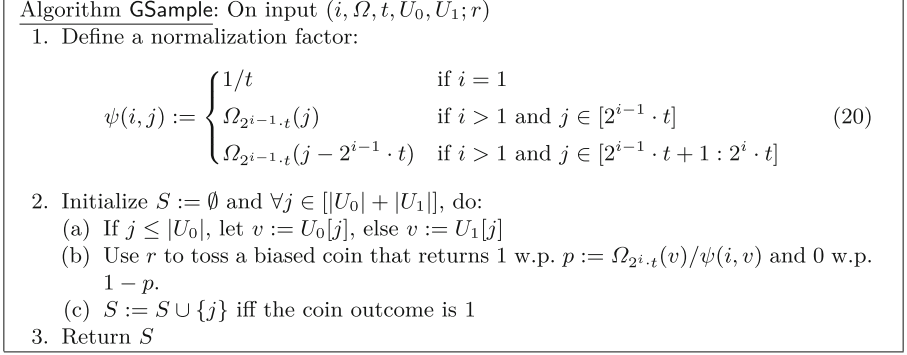
$\text{Inc}^{\tau(\cdot)}(1^\lambda, 1^{N'}, N, \pi, \chi)$ :

1. Parse  $\pi$  as  $(\phi_L, \mathcal{L}_{N,n-c}, \mathcal{I}_{N,n-c}, \mathcal{U}_{N,n-c}, U_{N,n-c})$ .
2. Compute  $N + N' = N''$  and check that  $N'' = 2^{n''}$  for some  $n'' \in \mathbb{N}$ .
3. Execute the algorithm  $\text{P}^{\tau(\cdot)}(1^\lambda, 1^{N''}, \chi)$  starting from step 2 with a slight change: traverse the graph  $G_{2^{n''}}$  starting from  $N + 1$ .

$\text{V}^{\tau(\cdot)}(1^\lambda, 1^N, \pi, \chi)$ :

1. Parse  $\pi$  as  $(\phi_L, \mathcal{L}_{N,n-c}, \mathcal{I}_{N,n-c}, \mathcal{U}_{N,n-c}, U_{N,n-c})$ .
2. If  $|\mathcal{L}_{N,n-c}| \neq |U_{N,n-c}|$ , return 0.
3. Let  $s := |\mathcal{L}_{N,n-c}|$ . For all  $i \in [s]$ :
  - (a) Parse  $\mathcal{L}_{N,n-c}[i]$  as  $y_i = (y_{i_1}, \dots, y_{i_k})$ , for some  $k \in \mathbb{N}$ .
  - (b)  $b_i := 1$  iff the following hold

- i.  $\tau_\ell(y_{i_k}) = \phi_L$
  - ii. **Consistent**( $y_i$ ) = 1 where **Consistent** is as in Def. 4,
  - iii. **GCheck**( $\mathcal{L}_{N,n-c}[i], \mathcal{I}_{N,n-c}[i], \mathcal{U}_{N,n-c}[i], U_{N,n-c}, 0, N, n-c$ ) = 1, where the algorithm **GCheck** is described in Fig. 5, and
4. Return  $\bigwedge_{i=1}^s b_i$



**Fig. 4.** Algorithm **GSample**

### 6.3 Theorem Statement and Proof Outline

The main difference in the constructions of Sect. 5 and 6 is that we apply the new on-the-fly sampling technique as employed in **GSample** in Fig. 4 to two sets  $U_0$  and  $U_1$  to obtain  $W$ . A second difference is that, while in Sect. 5, the sampling sets  $U_0, U_1$  are implicitly defined, and hence are assumed to be given to the verifier  $V$ , they are explicitly provided by the prover  $P$  in the main construction. Modulo these differences, the two constructions are essentially identical. In Theorem 3 below, the bound  $q \cdot \epsilon_0$  comes from analyzing our on-the-fly sampling, and the bound  $q \cdot \epsilon_1$  comes from the analysis of a new scenario where a malicious prover commits to sampling sets that are maliciously chosen.

**Theorem 3.** *Let  $\Pi := (P^{\tau(\cdot)}, V^{\tau(\cdot)}, \text{Inc}^{\tau(\cdot)})$  be as in Sect. 6 and  $q$  be an upper bound on the total number of queries to  $\tau(\cdot)$ , then  $\Pi$  is an  $(\alpha, \epsilon)$ -soundness incremental PoSW scheme, as per Def. 7, with any  $\alpha \in (0, 1]$  and*

$$\epsilon = \frac{1 + q^2}{2^\lambda} + \frac{q(q-1)}{2^{\lambda+1}} + q \cdot (\epsilon_0 + \epsilon_1) \quad (21)$$

where  $\epsilon_0, \epsilon_1 \in (0, 1)$  depend on the underlying  $t$ -incrementally sampleable weight distribution  $\Omega$ . For uniform  $\Omega$ , we have

$$\epsilon_0 \leq e^{-\frac{t \cdot (1-\alpha)^2 \cdot \zeta^{2(n-c)+4}}{(n-c)^2 \cdot (2-\zeta)^{2(n-c)+3}}} \quad \text{and} \quad \epsilon_1 \leq 2^{-\zeta \cdot t} .$$

In the full version, we give concrete  $\epsilon_0, \epsilon_1$  for any  $t$ -incrementally sampleable  $\Omega$ , as well as concrete upper bounds for the **SNACK** distribution.

Algorithm GCheck

On input a path  $\text{Path}$ , a list of indices  $\text{ind}$ , a list of lists of nodes  $\mathcal{U}$ , a list of nodes  $U$ , a starting node  $\mathbf{s}$ , an ending node  $\mathbf{e}$ , and a recursion index  $\mathbf{d}$ :

1. Parse  $\text{Path} := (y_1, \dots, y_k)$  and  $\text{ind} := (i_0, i_1, \dots, i_d) \in \mathbb{N}^{d+1}$
2. Parse  $\mathcal{U} := (U_{0,0}, U_{1,0}, U_{0,1}, U_{1,1}, \dots, U_{0,d-1}, U_{1,d-1})$
3. Compute  $r_{\mathbf{e},\mathbf{d}} := \tau_r(\mathbf{e}, \mathbf{d}, \tau_\ell(y_k), U_{0,d-1}, U_{1,d-1})$  and  $S_{\mathbf{e},\mathbf{d}} := \text{GSample}(d, \Omega, t, U_{0,d-1}, U_{1,d-1}; r_{\mathbf{e},\mathbf{d}})$
4. Let  $h_0 = |U_{0,d-1}|$  and  $h_1 = |U_{1,d-1}|$
5. Let  $w_0 = \Omega_{2^{d-1},t}(U_{0,d-1})$  and  $w_1 = \Omega_{2^{d-1},t}(U_{1,d-1})$
6. Let  $U_{0,d-1} || U_{1,d-1}$  be the list of nodes obtained by concatenating lists  $U_{0,d-1}$  and  $U_{1,d-1}$ . Let  $U^* := \{(U_{0,d-1} || U_{1,d-1})[s] : s \in S_{\mathbf{e},\mathbf{d}}\}$ . If  $U \neq U^*$ , return 0
7. Write

$$a := \begin{cases} 0 & \text{if } S_{\mathbf{e},\mathbf{d}}[i_d] \leq h_0 \\ 1 & \text{if } S_{\mathbf{e},\mathbf{d}}[i_d] > h_0 \end{cases} \quad \text{and} \quad b := S_{\mathbf{e},\mathbf{d}}[i_d] - h_0 \cdot a.$$

8. Define  $\mathbf{m} := (\mathbf{e} - \mathbf{s})/2$
9. If  $\mathbf{d} = 1$ , do the following:
  - Compute the challenge  $c$  corresponding to  $\text{Path}$
  - Compute  $j$  such that  $c \in [j \cdot t + 1 : (j+1) \cdot t]$
  - If  $a = 0$ , return 1 iff  $\mathbf{s} + b = c$  and  $U_{0,0} = [j \cdot t + 1 : (j+1) \cdot t]$  and  $U_{1,0} = [(j+1) \cdot t + 1 : (j+2) \cdot t]$
  - If  $a = 1$ , return 1 iff  $\mathbf{s} + \mathbf{m} + b = c$  and  $U_{0,0} = [(j-1) \cdot t + 1 : j \cdot t]$  and  $U_{1,0} = [j \cdot t + 1 : (j+1) \cdot t]$
10. If  $\mathbf{d} > 1$ , do the following
  - If  $a = 0$ ,  $S_{\mathbf{e},\mathbf{d}}[i_d] = i_{d-1}$ ,  $(1 - \zeta) \cdot t \leq h_0, h_1 \leq (1 + \zeta) \cdot t$  and  $\omega_{\min,d-1,\Omega} \leq w_0, w_1 \leq \omega_{\max,d-1,\Omega}$   
 $\text{GCheck}((y_1, \dots, y_{k-1}), (i_1, \dots, i_{d-1}), (U_{0,0}, U_{1,0}, \dots, U_{0,d-2}, U_{1,d-2}), U_{0,d-1}, \mathbf{s}, \mathbf{s} + \mathbf{m}, \mathbf{d} - 1)$
  - If  $a = 1$ ,  $S_{\mathbf{e},\mathbf{d}}[i_d] = h_0 + i_{d-1}$ ,  $(1 - \zeta) \cdot t \leq h_0, h_1 \leq (1 + \zeta) \cdot t$  and  $\omega_{\min,d-1,\Omega} \leq w_0, w_1 \leq \omega_{\max,d-1,\Omega}$   
 $\text{GCheck}((y_2, \dots, y_k), (i_1, \dots, i_{d-1}), (U_{0,0}, U_{1,0}, \dots, U_{0,d-2}, U_{1,d-2}), U_{1,d-1}, \mathbf{s}, \mathbf{s} + \mathbf{m}, \mathbf{d} - 1)$
  - Return 0

**Fig. 5.** Description of the GCheck algorithm.

**Acknowledgements.** Parts of this work were done while the first author was at TU Wien and was supported by the Vienna Science and Technology Fund (WWTF)[10.47379/VRG18002]. The first author was also partially funded by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), by the Spanish Government under project PRODIGY (TED2021-132464B-I00), and by the Madrid Regional Government under project BLOQUES (S2018/TCS-4339). The last two projects are co-funded by European Union EIE, and NextGenerationEU/PRTR funds.

The research of the second author was in part funded by the Austrian Science Fund (FWF) and netidee SCIENCE grant P31621-N38 (PROFET).

## References





1. Abusalah, H., Alwen, J., Cohen, B., Khilko, D., Pietrzak, K., Reyzin, L.: Beyond Hellman's time-memory trade-offs with applications to proofs of space. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 357–379. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-70697-9\\_13](https://doi.org/10.1007/978-3-319-70697-9_13)
2. Abusalah, H., Fuchsbauer, G., Gazi, P., Klein, K.: Snacks: leveraging proofs of sequential work for blockchain light clients. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology - ASIACRYPT 2022–28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, 5–9 December 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13791, pp. 806–836. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-22963-3\\_27](https://doi.org/10.1007/978-3-031-22963-3_27)
3. Abusalah, H., Kamath, C., Klein, K., Pietrzak, K., Walter, M.: Reversible proofs of sequential work. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 277–291. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17656-3\\_10](https://doi.org/10.1007/978-3-030-17656-3_10)
4. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 757–788. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-96884-1\\_25](https://doi.org/10.1007/978-3-319-96884-1_25)
5. Cohen, B., Pietrzak, K.: Simple proofs of sequential work. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 451–467. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-78375-8\\_15](https://doi.org/10.1007/978-3-319-78375-8_15)
6. Cohen, B., Pietrzak, K.: The chia network blockchain, July 2019. <https://www.chia.net/assets/ChiaGreenPaper.pdf>
7. Döttling, N., Lai, R.W.F., Malavolta, G.: Incremental proofs of sequential work. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 292–323. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17656-3\\_11](https://doi.org/10.1007/978-3-030-17656-3_11)
8. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993). [https://doi.org/10.1007/3-540-48071-4\\_10](https://doi.org/10.1007/3-540-48071-4_10)
9. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 585–605. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48000-7\\_29](https://doi.org/10.1007/978-3-662-48000-7_29)
10. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 125–154. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45727-3\\_5](https://doi.org/10.1007/978-3-030-45727-3_5)
11. 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](https://doi.org/10.1007/3-540-47721-7_12)
12. Mahmoody, M., Moran, T., Vadhan, S.P.: Publicly verifiable proofs of sequential work. In: Kleinberg, R.D. (ed.) ITCS 2013, pp. 373–388. ACM, January 2013. <https://doi.org/10.1145/2422436.2422479>
13. Pietrzak, K.: Simple verifiable delay functions. In: Blum, A. (ed.) ITCS 2019, vol. 124, pp. 60:1–60:15. LIPIcs, January 2019. <https://doi.org/10.4230/LIPIcs.ITCS.2019.60>
14. Wesolowski, B.: Efficient verifiable delay functions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 379–407. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17659-4\\_13](https://doi.org/10.1007/978-3-030-17659-4_13)

# **(Zero-Knowledge) Proofs**





# Witness-Succinct Universally-Composable SNARKs

Chaya Ganesh<sup>1</sup>, Yashvanth Kondi<sup>2</sup>, Claudio Orlandi<sup>2</sup>, Mahak Pancholi<sup>2</sup>,  
Akira Takahashi<sup>3</sup>, and Daniel Tschudi<sup>4</sup>

<sup>1</sup> Indian Institute of Science, Bengaluru, India  
chaya@iisc.ac.in

<sup>2</sup> Aarhus University, Aarhus, Denmark  
{ykondi, orlandi, mahakp}@cs.au.dk

<sup>3</sup> University of Edinburgh, Edinburgh, Scotland  
takahashi.akira.58s@gmail.com

<sup>4</sup> Concordium, Zug, Switzerland  
dt@concordium.com

**Abstract.** Zero-knowledge Succinct Non-interactive ARguments of Knowledge (zkSNARKs) are becoming an increasingly fundamental tool in many real-world applications where the proof compactness is of the utmost importance, including blockchains. A proof of security for SNARKs in the Universal Composability (UC) framework (Canetti, FOCS'01) would rule out devastating malleability attacks. To retain security of SNARKs in the UC model, one must show their *simulation-extractability* such that the knowledge extractor is both *black-box* and *straight-line*, which would imply that proofs generated by honest provers are *non-malleable*. However, existing simulation-extractability results on SNARKs either lack some of these properties, or alternatively have to sacrifice *witness succinctness* to prove UC security.

In this paper, we provide a compiler lifting any simulation-extractable NIZKAoK into a UC-secure one in the global random oracle model, importantly, while preserving the same level of witness succinctness. Combining this with existing zkSNARKs, we achieve, to the best of our knowledge, the first zkSNARKs simultaneously achieving UC-security and constant sized proofs.

---

The authors would like to thank abhi shelat for helpful discussions about an early version of this work. We thank anonymous reviewers of Eurocrypt 2023 for valuable comments and suggestions.

The work described in this paper has received funding from: the Concordium Blockchain Research Center, Aarhus University, Denmark; the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM); the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC); Core Research Grant CRG/2020/004488, SERB, Department of Science and Technology; Infosys Young Investigator Award, Infosys Foundation, Bangalore; the Protocol Labs Research Grant Program PL-RGP1-2021-064.

© International Association for Cryptologic Research 2023

C. Hazay and M. Stam (Eds.): EUROCRYPT 2023, LNCS 14005, pp. 315–346, 2023.

[https://doi.org/10.1007/978-3-031-30617-4\\_11](https://doi.org/10.1007/978-3-031-30617-4_11)

## 1 Introduction

**The UC framework and UC Secure NIZKs.** The Universal Composability (UC) framework [28] allows for the modular design and analysis of complex cryptographic protocols, and guarantees security in the presence of arbitrarily many sessions running concurrently. The *environment*  $\mathcal{Z}$  (representing everything that is external to the execution of the protocol of interest) interacts with the protocol, at the conclusion of which it outputs a decision bit, indicating whether it thinks it has interacted with a “real-life” adversary  $\mathcal{A}$  and parties running the protocol, or with an “ideal-process” adversary (or *simulator*)  $\text{Sim}$  and parties accessing the so-called *ideal functionality*  $\mathcal{F}$  specifying the ideal outcome of a given protocol.

This paper focuses on non-interactive zero-knowledge proofs (NIZK) [16, 17] in the UC framework. In the standalone setting, security of NIZKs is guaranteed by showing standard properties separately such as completeness, zero-knowledge, and (knowledge) soundness under some setup assumptions, like a common reference string (CRS) or the Random Oracle Model (ROM). However, several restrictions and stronger properties come into play once the NIZK functionality is to be realized in the UC model. A common methodology to design NIZKs in the ROM is to start with an interactive argument which is proven ZK/knowledge sound, and then compile this interactive argument into a non-interactive proof. This means that NIZKs that are proven secure using rewinding (either for ZK or for extraction) are at odds with UC, because the environment  $\mathcal{Z}$  is an interactive distinguisher between the real execution protocol and the ideal process, and therefore a simulator  $\text{Sim}$  in the security proof cannot rewind  $\mathcal{Z}$ . Thus, *straight-line* simulation and extraction are required for a NIZK to be UC secure. Informally, a proof system is straight-line extractable if one can efficiently extract a valid witness without interacting with any successful prover. On top of extraction being straight-line, by definition, UC simulators must be able to obtain a witness having only *black-box* access to  $\mathcal{Z}$ , i.e., without knowing the concrete code of  $\mathcal{Z}$ .

Another important ingredient to realize UC security is *non-malleability* (NM) [37], which is often referred to as *simulation-extractability* in context of UC (NI)ZK [73, 35, 72, 46, 56, 38]. Essentially, a malleability attack allows an adversary to maul existing proofs observed during the protocol execution, and to forge a proof on some statement for which they do not know the corresponding witness. Preventing such attacks is crucial in the UC model: as  $\mathcal{Z}$  may ask uncorrupted provers or simulator to produce proofs on arbitrary statement-witness pairs, the ability to maul such proofs will induce the simulation to fail (i.e.,  $\text{Sim}$  fails to extract a witness) and thus helps  $\mathcal{Z}$  distinguish the real execution from the ideal one. The non-malleable NIZK construction of [35] was shown to be UC secure in [31]. Subsequently, [54, 50] constructed UC secure NIZKs in the presence of adaptive adversaries, and [50] proved that simulation-extractability is necessary for UC. In sum, black-box extraction (BBE), straight-line extraction (SLE) and simulation-extractability (SIMEXT) are the properties a NIZK must satisfy in order to be UC secure.

We now discuss UC security for SNARKs (*succinct* non-interactive arguments<sup>1</sup> of knowledge) where the communication is sublinear (ideally polylogarithmic or constant<sup>2</sup>) in the size of the non-deterministic witness used to verify the relation. A SNARK is *circuit-succinct* if the proof size is sublinear only in the size of the circuit representing the statement; if it is sublinear in the length of the witness too, it is *witness-succinct*. Many SNARK constructions in the literature rely on *knowledge assumptions* to prove witness extraction, i.e. their extractors rely on examining the concrete code of the adversary in order to extract a witness. As discussed earlier, this is a barrier to achieving UC security, as simulation in the UC framework can not depend on the code of the environment.

One simple folklore method to obtain UC-secure circuit-succinct NIZK given a SNARG (a SNARK that only guarantees soundness, not proof-of-knowledge) and a (perfectly correct) public key encryption scheme is the following: a public key  $pk$  serves as a common reference string, given which the prover computes a ciphertext  $ct$  to encrypt the witness  $w$  under randomness  $r$ . The prover then computes a SNARG  $\pi$  that proves that the message encrypted by  $ct$  is indeed a witness to the statement, and outputs  $(ct, \pi)$ . This tuple now constitutes a straight-line extractable NIZK, as the extractor (given  $sk$ ) can simply decrypt  $ct$  to obtain  $w$ —intuitively this  $w$  must be a valid witness since  $ct$  is a perfectly binding commitment to  $w$ , and so if  $w$  is not a valid witness then  $\pi$  would be proving a false theorem. Notice that this proof additionally inherits the *circuit succinctness* property of the SNARG, as  $ct$  is of size  $O(|w|)$  and  $\pi$  is the SNARG itself. This approach was described by De Santis et al. [35] in the context of lifting ordinary NIZK to simulation-sound NIZK, and implemented as part of the  $C\emptyset C\emptyset$  framework for circuit-succinct UC NIZK by Kosba et al. [63], with optimizations for concrete efficiency using the state-of-the-art SNARKs at the time.  $C\emptyset C\emptyset$  further proposed an optimized method to obtain non-malleability, by additionally proving that the encrypted string is a valid signature on the statement. Putting all these features together,  $C\emptyset C\emptyset$  serves as the first *generic* UC lifting compiler preserving circuit succinctness.

A major limitation of this technique is that it is inherently limited to producing proofs that are at least as large as the witness, by virtue of the witness having to be ‘decryptable’ from the ciphertext. Constructing *witness-succinct* proofs that enjoy black-box straight-line extraction appears to require a fundamentally different approach. Indeed, Kosba et al. remarked that there is “no known UC-secure zero-knowledge proof construction that is circuit *and* witness-succinct, even under non-standard assumptions” [63, pg. 2], and left open the question of whether such an object is even feasible to construct. Given this, one may ask:

*Is it possible to obtain UC-secure witness-succinct NIZKs  
under well-studied setup assumptions?*

<sup>1</sup> Argument systems are proofs where soundness is computational. For proofs to be shorter than the length of the witness, restricting to arguments is necessary [48, 49].

<sup>2</sup> Polynomial only in the security parameter.

The requirement of “well-studied” setup assumptions is meant to capture those forms of setup that have generally accepted realizations. In this work, we consider the *common reference string* (CRS) model, and the *random oracle model* (ROM) to fall within the scope of well-studied setup. For SNARKs in particular, there is already established infrastructure to generate the CRSs required (via so called “powers of tau” ceremonies implemented by major blockchains such as ZCash, FileCoin, etc. [21]). There are also established heuristics to instantiate the ROM in practice with carefully chosen hash functions, and the ROM itself is arguably amongst the oldest and most comprehensively studied idealized models [10].

**Models we do *not* Consider.** Several SNARK constructions are known to be secure with non-black-box extraction under knowledge assumptions, or in idealized models such as the Generic Group Model (GGM) or Algebraic Group Model (AGM). The UC-AGM framework [1] allows to model the AGM and algebraic adversaries in a composable fashion. However, doing so requires the use of algebraic environments making it incompatible with standard UC. The other related alternative model is considered in [60] where they formally define the concept of *knowledge-respecting distinguishing environments*, enabling the usage of primitives relying on knowledge assumptions in larger protocols. However, their entire formalization is built on top of a different compositional framework [68] than UC. Similar to the UC-AGM framework, distinguishers in their model are globally assumed to explain how they computed each knowledge-implying object they output, making themselves weaker than environments in the standard UC.

**Succinct Arguments of Knowledge with a CRS Alone.** Folklore has long held that NIZKs in the CRS model with black-box straight-line extraction cannot be witness-succinct, as the witness must be ‘decryptable’ from the proof string as in the simple approach described earlier. Indeed, all pairing based efficient SNARKs that are witness-succinct in the standard model with a CRS (like [47, 70]) are not black-box extractable<sup>3</sup>. The intuition is that for a language whose witnesses have enough entropy, an argument that is too “short” cannot contain enough information about a witness: this makes extraction impossible for an extractor that does not have any additional power, like access to the prover’s randomness (like in non-black-box extractors) or the ability to rewind the prover (like in interactive arguments and resulting NIZKs compiled in the ROM). We refer the reader to the recent work of Campanelli et al. [27] for a formal treatment of this. Given that black-box extraction is necessary for UC security, we consider it justified to consider UC security in the ROM in light of this impossibility.

**Succinct Arguments of Knowledge in the ROM.** There are several witness-succinct proof systems in the ROM in the literature such as the classical Probabilistically Checkable Proofs (PCP) based approach of Kilian [61], Micali’s CS proofs [69], and the recent works on Interactive Oracle Proofs [15]. However to our knowledge, there are no witness-succinct proof systems in the ROM that

---

<sup>3</sup> Pairing based constructions like PLONK, Sonic, Marlin are not black-box extractable as well, but they are also in the ROM in addition to requiring a CRS.

**Table 1.** Known properties of existing (witness-succinct) zkSNARKs compared to example instantiation of our compilation. “BBE” stands for black-box knowledge extraction; “SLE” for straight-line knowledge extractor; “SIMEXT” for simulation-extractability. We say a proof system is “transparent” if no trusted generation of CRS is required. Note that the assumptions for the last row are derived from an example instantiation of [2, Theorem 4] where they adapt [52] as an underlying SNARK.

Scheme	Assumption	Model	Transparent	BBE	SLE	SIMEXT
STARK [12]	ROM	ROM	✓	✓	✓	unknown
Aurora [14]	ROM	ROM	✓	✓	✓	unknown
RedShift [59]	ROM	ROM	✓	✓	✓	unknown
Bulletproofs [22]	DLOG	ROM	✓	✓	✗	✓ [45]
SONIC [67]	AGM & $q$ -DLOG	CRS & ROM	✗	✗	✓	✓ [42]
PLONK [41]	AGM	CRS & ROM	✗	✗	✓	✓ [42]
Marlin [33]	AGM	CRS & ROM	✗	✗	✓	✓ [42]
Groth16 [51]	GGM	CRS(& ROM for NM)	✗	✗	✓	✓ [20]
Groth-Maller [53]	XPKE & Poly	CRS	✗	✗	✓	✓
LAMASSU [2]	$q$ -MC & $q$ -MK & BDH & DL	CRS	✗	✗	✓	✓
Ours + [53] + [57]	XPKE & Poly & SDH	CRS & GROM	✗	✓	✓	✓
Ours + [2] + [57]	$q$ -MC & $q$ -MK & BDH & DL & SDH	CRS & GROM	✗	✓	✓	✓

have been formally analyzed in the UC framework. While some of these constructions [69, 3, 11] are black-box straight-line extractable, simulation-extractability of these has not been shown. SNARKs in the ROM that are logarithmic in the statement and witness size are known from conservative computational assumptions such as the hardness of computing discrete logarithms [19, 22] in the standalone setting. Bulletproofs [22] are known to be simulation-extractable, but currently only in the AGM+ROM [44] or in the ROM with rewinding [45]. If a CRS is assumed in addition to ROM, then constructions like PLONK, Sonic, and Marlin also provide constant sized (polynomial only in the security parameter) proofs, but their simulation-extractability is only shown in AGM+ROM [42]. We indicate these properties of existing SNARKs in Table 1. Given this state of affairs, we can refine our earlier question to the following:

*Is it possible to obtain UC-secure NIZKs with constant size proofs in the **random oracle model**?*

**Our Results.** In this work, we answer the above question in the affirmative. In particular, we give a compiler (in the ROM) that lifts any SNARK from non-black-box to black-box straight-line extraction, with *constant* (i.e.  $O_\lambda(1)$ ) overhead.

**Theorem 1.1.** (Informal) *Given a non-black-box simulation-extractable zkSNARK  $\Pi_{\mathcal{R}}$  for a relation  $\mathcal{R}$  and a succinct polynomial commitment scheme, there exists a UC-secure, witness-succinct zkSNARK  $\Pi_{UC-\mathcal{R}}$  in the (global random oracle ( $\mathcal{G}_{RO}$ ), local setup ( $\mathcal{F}_{Setup}$ ))-hybrid model, where  $\mathcal{G}_{RO}$  is observable but non-programmable as in [30] and  $\mathcal{F}_{Setup}$  models the setup required by the original zkSNARK  $\Pi_{\mathcal{R}}$  (e.g., a trusted CRS generator or the local random oracle).*

Plugging well-known SNARKs such as [53, 2] into our compiler gives us as a corollary the first constant sized UC NIZKs in the  $(\mathcal{G}_{\text{RO}}, \mathcal{F}_{\text{crs}})$ -hybrid model, from pairings under knowledge assumptions.

**Remarks.** There are a few qualifications to our main theorem:

- *Knowledge Assumptions:* Any output NIZK produced via our compiler inherits the knowledge assumptions used by the input SNARK. However, as knowledge assumptions cannot be used directly in the UC framework (as simulation cannot depend on the environment), the extraction strategy for our compiled SNARK does not involve invoking the non-black-box extractor of the input SNARK. Intuitively, we only make use of the input SNARK’s non-black-box extractor to argue the indistinguishability of intermediate hybrid experiments (which can depend on the environment).
- *Unique Proofs:* Our compiler requires polynomial commitments that support a new ‘unique proof’ property, i.e. it is hard for an adversary to produce two distinct proofs for the same evaluation point. This is in fact an analogous notion to *unique response* defined for ROM-based NIZK proofs to be simulation-extractable [38, 44]. Although this is not a standard property in the stand-alone setting, we show that it is a natural feature of common polynomial commitment schemes such as KZG [57].

## 1.1 Technical Overview

We begin with the observation that most SNARKs already have *straight-line zero-knowledge simulators*—the verifier of a non-interactive object has no secrets and so there is nothing to be gained by rewinding or looking at its code—and therefore simulating an honest prover’s SNARK string in the UC context is straightforward. Moreover, a plethora of work suggest that many concretely efficient SNARKs are already *simulation extractable* (see Table 1). The barrier to using existing SNARKs in the UC context is that the only known extractors require either looking into the code of the prover (i.e. non-black-box extraction) or rewinding the prover. Neither of these extraction techniques can be directly used within the UC framework, as the simulator in the UC experiment can not rewind the environment, nor depend on its code.

Previous works have recognized the fact that even though simulation must be straight-line in the UC framework, their proofs of indistinguishability can make use of arguments that involve rewinding the environment [36, 25]. The underlying principle is that even though the environment can not be rewound during simulation for the UC experiment, rewinding the environment can still be helpful as an analytical tool, for example in generating intermediate hybrid distributions between the real and ideal experiments. To our knowledge, this principle has not been applied to the case of *non-black-box* simulation, i.e. generating intermediate hybrid distributions using the code of the environment.

Our insight is that the existence of a non-black-box extractor guarantees that in order to produce a SNARK, the environment must fundamentally ‘know’ a witness—lifting the SNARK to a UC NIZK is then a matter of forcing the

environment to use this knowledge. We describe below how we leverage this insight, by incrementally building upon the simple approach described earlier.

**Commitments Instead of Encryption.** Recall that the simple approach—where a proof consists of ciphertext  $\text{ct}$  and proof  $\pi$  that  $\text{ct}$  encrypts a witness—is bottlenecked by the ciphertext having to be ‘decryptable’, which means that  $|\text{ct}| \in \Omega(|w|)$ . If we relax the decryptability requirement, we can have  $\text{ct}$  be a *commitment* instead. This is helpful, because commitments can be independent of the size of the message committed, and therefore succinct. Obtaining the witness from  $\text{ct}$  now becomes a matter of extracting a committed message rather than simply decrypting a ciphertext, and forms the core of the technical challenge.

**Core Tool: Succinct, Provable, Straight-Line Extractable Commitments.** Straight-line extractable commitments are typically straightforward to construct in the random oracle model—simply computing  $H(w, r)$  to commit to  $w$  with randomness  $r$  suffices [71, 25]. However  $H$  must be a random oracle to enable straight-line extraction, meaning that one cannot prove statements about its input. This is an issue as we need to prove that  $w$  committed to in  $\text{ct}$  is indeed a valid witness. This issue can be solved by assuming that since  $H$  is instantiated with a concrete hash function, it will have a circuit representation (as is common in the literature on recursive SNARKs [23, 34]) however we wish to avoid such heuristics.

We must therefore construct a ‘provable’ commitment scheme, i.e. one that has a meaningful circuit representation while also supporting straight-line extraction of the committed message. Our methodology for designing such a commitment involves two parts  $(\text{cm}, \pi_{\text{cm}})$ , where  $\text{cm}$  is a commitment string output by a standard model commitment algorithm  $\text{cm}$ , and  $\pi_{\text{cm}}$  is a straight-line extractable proof of knowledge of its opening—notice that now it is meaningful to prove via a SNARK that  $\text{cm}$  is a commitment to a valid witness, as  $\text{cm}$  is a standard model algorithm. Since it is straightforward to achieve  $|\text{cm}| \in O_\lambda(1)$ , we will focus on the design of  $\pi_{\text{cm}}$ .

Like much of the SNARK literature, in constructing  $\pi_{\text{cm}}$  we leverage the fact that arithmetization is conducive to succinct proofs. In particular, we instruct the prover to encode the witness  $w$  as the coefficients of a polynomial  $f(x)$ , and commit to  $f$  within  $\text{cm}$  (rather than committing to  $w$  directly). Assuming a prime  $q \in \omega(\text{poly}(\lambda))$  is a parameter of the scheme, and  $d \in \mathbb{Z}$  a parameter of the statement,  $w$  is interpreted as a vector  $w \in \mathbb{F}_q^d$  that characterize the coefficients of the degree<sup>4</sup>  $d - 1$  polynomial  $f \in \mathbb{F}_q[X]$ . Our straight-line extraction strategy will be to ensure that the prover queries at least  $d$  evaluations of  $f$  to the random oracle (i.e. enough to reconstruct  $w$ ), by having the verifier check a subset of the evaluations. Importantly, this validation of  $f$  can be performed succinctly; the verifier need only query  $O_\lambda(1)$  evaluations of  $f$ , and each evaluation can be authenticated at  $O_\lambda(1)$  cost. We sketch our ideas behind these principles below.

<sup>4</sup> We remark that the actual compiler needs to inflate the degree according to the number of revealed evaluations in order to retain zero-knowledge, but we omit this technicality here for ease of exposition.

$O_\lambda(1)$  **Verifier Queries:** The prover first evaluates  $f$  at  $n$  points and commits to each  $\{f(i)\}_{i \in [n]}$ . The prover is then instructed to reveal  $r$  of the committed evaluations—which are checked for correctness—to guarantee that the commitments contain at least  $d - 1$  correct evaluations in total, with overwhelming probability. Assuming that  $r \in [n]$  is chosen at random, the parameters can be fixed so that  $r \in O_\lambda(1)$ , due to the following rough analysis: the best adversarial assignment (for a cheating prover) of the  $n$  committed evaluations consists of only  $d - 1$  correct (and  $n - d + 1$  ‘junk’) ones, to maximize the number of subsets of size  $r$  that will satisfy a verifier—i.e.  $\binom{d-1}{r}$ . The total number of possible subsets that the verifier could query is  $\binom{n}{r}$ , which brings the probability of success of the best possible cheating strategy to:

$$\frac{\binom{d-1}{r}}{\binom{n}{r}} \approx \frac{d^r/r!}{n^r/r!} = \left(\frac{d}{n}\right)^r$$

Now if we fix  $r$  as say,  $\lambda$  (so that  $r \in O_\lambda(1)$ ), notice that for any  $d \in \mathbb{Z}$  the above quantity can be upper bounded by  $2^{-\lambda}$  by setting  $n \approx 2d$ . In general, as long as  $r \in \Omega(\lambda/\log \lambda)$ , the same upper bound can be achieved with  $n \in \text{poly}(d, \lambda)$ .

**Authenticating Evaluation Openings at  $O_\lambda(1)$  Cost via Fischlin’s Technique [39]:** We framed our description above in a PCP-like model, where the prover writes down  $n$  evaluations of  $f$ , of which the verifier queries and checks  $r$  of them. As  $n$  is clearly not witness-succinct, we need a method by which the prover can commit to the  $n$  evaluations, and succinctly reveal  $r$  of them upon request. In the PCP/IOP literature [69, 15], it is common to use Merkle trees for this task; they provide  $O_\lambda(1)$  sized commitments with  $r$  short ( $O_\lambda(\log n)$  sized) openings, and even natively support straight-line extraction. This follows a ‘cut-and-choose’ paradigm, where the prover commits to  $n$  objects, and the verifier checks  $r$  of them in order to guarantee that a total of at least  $d$  of the committed objects are ‘good’. However the  $O_\lambda(\log n)$  sized evaluation opening is a deal breaker (in the context of achieving  $O_\lambda(1)$ -sized proofs) as it grows—albeit slowly—with the witness size, and appears to be a fundamental hurdle with such techniques.

In the context of compiling  $\Sigma$ -protocols to NIZKs with straight-line extraction, Fischlin [39] presented a technique based on *proofs of work* that shed the  $O_\lambda(\log n)$  cost of Merkle tree openings when checking the validity of a subset of committed objects. At a very high level, Fischlin’s technique emulates the combinatorial properties of the cut-and-choose approach, without the logistics of providing explicit commitments/openings. Fischlin’s idea is that rather than challenging the prover to reveal a (randomly chosen)  $r$ -sized subset of some committed  $x_i$  values, the prover is challenged to provide any  $r$  values  $\{x_i\}_{i \in [r]}$  such that  $H(x_i) = 0$  for each  $i$ , where  $H$  is a random oracle. This forces the prover to query multiple ‘good’  $x_i$  values to  $H$  before finding  $r$  of them that hash to the zero string, and no explicit decommitment information is necessary.

Applying Fischlin’s technique to our setting yields a protocol of the following form. Upon fixing  $\text{cm}$ , for each  $i \in [r]$ : (1) the prover computes  $\pi_{\text{cm}}^{(i)} = (z_i, f(z_i))$  with uniform  $z_i$  and the corresponding *evaluation proof*  $\pi_{\text{ev}}^{(i)}$  that ensures the



polynomial  $f$  committed to in  $\text{cm}$  has been correctly evaluated at  $z_i$ , and (2) store  $(\pi_{\text{cm}}^{(i)}, \pi_{\text{ev}}^{(i)})$  and go to the next iteration if  $H(\text{cm}, i, \pi_{\text{cm}}^{(i)}, \pi_{\text{ev}}^{(i)}) = 0$  for a random oracle  $H$  with  $b$ -bit outputs, and go to step (1) otherwise. Thanks to the evaluation proof,  $\pi_{\text{cm}}^{(i)}$  is tied to a given commitment  $\text{cm}$ . In practice, succinct evaluation proof can be easily implemented by naively invoking the underlying SNARK prover<sup>5</sup> or by instantiating  $\text{cm}$  with a dedicated *polynomial commitment scheme* such as [57], which usually minimizes the overhead in prover’s work. Computing such a proof is easy for an honest prover, via rejection-sampling with random  $(z_i, f(z_i))$  values until  $r$  of them that hash to zero are found. As for an adversarial prover  $P^*$ , the aim is to produce an accepting proof—by finding  $r$  pre-images of 0—with  $d - 1$  or fewer queries to the random oracle. As a loose upper bound, the probability that  $P^*$  finds  $r$  successes within  $d - 1$  queries is at most the probability that for every  $i \in [r]$ ,  $P^*$  is able to find  $H(\text{cm}, i, \cdot) = 0$  within  $d - 1$  queries. For any given  $i$ , the probability that  $P^*$  finds  $H(\text{cm}, i, \cdot) = 0$  within  $d$  queries is at most  $d/2^b$ ; therefore the probability that  $P^*$  finds  $H(\text{cm}, i, \cdot) = 0$  within  $d - 1$  queries for every  $i \in [r]$  simultaneously is at most  $(d/2^b)^r = 2^{-r(b - \log d)}$ . The proof sketch here are implicitly assuming that a valid evaluation proof is determined *uniquely* once  $\text{cm}$ ,  $z_i$ , and  $f(z_i)$  are fixed. Our formal analysis accounts for this subtlety and we show that [57] indeed satisfies this property.

Assuming that  $r = \lambda \in O_\lambda(1)$ , the above quantity is bounded by  $2^{-\lambda}$  when  $b = 1 + \log d \in O_\lambda(\log d)$ . The prover’s work is in expectation  $2^b \cdot r = 2^{1 + \log d} \cdot \lambda$  which is in  $\text{poly}(\lambda)$  as well as  $O_\lambda(d)$ , i.e. it scales linearly in the witness size. Of course better parameters are possible;  $r$  can be improved by up to a log factor, as we explore later in the ‘succinctness’ component of the proof of Theorem 3.1.

**Putting it Together:** The prover produces an  $O_\lambda(1)$ -sized standard model commitment  $\text{cm}$  to a degree  $d$  polynomial  $f$  that encodes the witness, and proves knowledge of its opening via  $\pi_{\text{cm}} = (\pi_{\text{cm}}^{(i)})_{i \in [r]}$ —this proof is at the heart of forcing the environment to use the witness within the context of the protocol. The proof  $\pi_{\text{cm}}$  requires the prover to ‘work’ to find  $r \in O_\lambda(1)$  pre-images of 0 for random oracle  $H$ , where each pre-image is an evaluation of  $f$ . The parameters for this proof-of-work are set so that (except with negligible probability) the prover queries more than  $d - 1$  evaluations of  $f$  in its effort to find these  $r$  pre-images of zero. Reading these  $d$  evaluations of  $f$  allows an extractor to reconstruct  $f$ —which is an opening to  $\text{cm}$ . Finally, the prover gives a SNARK  $\pi$  to prove that it knows an opening to  $\text{cm}$  that is the witness to a public statement (through a suitable witness-polynomial encoding function  $\text{Enc}$ ). If one were to hypothetically run the non-black-box SNARK extractor on the environment at this point, the opening to  $\text{cm}$  that it finds should be *exactly the same* as the  $f$  reconstructed via the extractor of  $\pi_{\text{cm}}$ ; if not, then one would obtain two openings to  $\text{cm}$ , in contradiction of the binding property of the commitment scheme. Therefore,

<sup>5</sup> For this alternative instantiation, one must use a de-randomized version of the underlying SNARK to obtain the unique proof property, as also required by our main compiler.

any knowledge that the environment uses in the production of  $\pi$ —perhaps even outside the protocol—is extracted in a black-box, straight-line fashion via  $\pi_{\text{cm}}$  within the context of the protocol.

## 1.2 Related Work

**Straight-Line Extraction.** Our UC-lifting technique is inspired by Fischlin’s transform [39] based on Proof-of-Work. Kondi and Shelat [62] gave an analysis for using Fischlin’s transformation for *compressing* proofs in the context of signature aggregation, and showed how randomizing Fischlin’s technique is conducive to zero-knowledge. Very recently, Lysyanskaya and Rosenbloom [66, 65] present compilers lifting  $\Sigma$ -protocols to UC-secure (adaptive) NIZKPoK in the global ROM, where the straight-line extraction is realized via Fischlin’s transform. Canetti, Sarkar, and Wang [32] realized *triply adaptive* UC-secure NIZK using a straight-line extractable commitment in the CRS model. Pass [71] described a generic way to turn  $\Sigma$ -protocols with special soundness into straight-line extractable proof systems using RO-based commitment. The technique is somewhat analogous to the verifiable encryption of Camenisch and Damgård [24] where the commitment is instantiated using public-key encryption and thus SLE holds in the CRS model (where the decryption key serves as a private extraction key for the knowledge extractor). The transform of Unruh [74] extended [71] to retain security against an adversary making superposition queries to the RO (the so-called *quantum random oracle model*). Recently, Katsumata [58] showed an efficient SLE transform in the QROM tailored to lattice-based ZK proofs.

**Lifting Transformations.** Techniques for generically adding black-box simulation extractability to any NIZK were first shown in the works of [73, 35, 50], optimized in the  $C\emptyset C\emptyset$  framework [63], and tailored to Groth16 in [5, 6]. These techniques augment the relation to an OR language and the trapdoor for one of the OR clauses is used by the ZK simulator. Extractability is obtained by encrypting the witness under a public key that is part of the CRS and additionally proving correct encryption. The LAMASSU [2] framework extends the  $C\emptyset C\emptyset$  lifting technique to work with updatable SNARKs giving a generic compiler from updatable CRS SNARKS to SE SNARKs. TIRAMISU [9] builds on these frameworks to additionally lift SNARKs into black-box simulation extractable ones. However, all these lifting transformations yield SNARKs where one of either witness succinctness or blackbox extraction is lost, unlike our compiler. There are works on lifting specific SNARKs into SE; the work of Groth and Maller [53] presents an SE SNARK, but the simulation extractability is non-black-box. There is a line of work on analysing the simulation extractability [20, 7, 8] of Groth16; all of these are in idealized models like GGM/AGM, in addition to ROM.

## 2 Preliminaries

**Notations.** For positive integers  $a$  and  $b$  such that  $a < b$  we use the integer interval notation  $[a, b]$  to denote  $\{a, a + 1, \dots, b\}$ . We also use  $[b]$  as shorthand

for  $[1, b]$ . If  $S$  is a set we write  $s \leftarrow_s S$  to indicate sampling  $s$  from the uniform distribution defined over  $S$ ; if  $\mathcal{A}$  is a randomized (resp. deterministic) algorithm we write  $s \leftarrow \mathcal{A}$  (resp.  $s := \mathcal{A}$ ) to indicate assigning an output from  $\mathcal{A}$  to  $s$ . The security parameter  $\lambda$  is  $1^\lambda$  in unary. A function  $f(\lambda)$  is said to be *negligible in  $\lambda$*  if for any polynomial  $\text{poly}(\lambda)$  it holds that  $f(\lambda) < 1/\text{poly}(\lambda)$  for sufficiently large  $\lambda > 0$ . We write “ $f(\lambda) < \text{negl}(\lambda)$ ” to indicate  $f(\lambda)$  is negligible in  $\lambda$ .  $\mathbb{F}[X]$  denotes polynomials over a finite field  $\mathbb{F}$ . For an integer  $d \geq 1$ ,  $\mathbb{F}_{<d}[X] \subseteq \mathbb{F}[X]$  denotes polynomials of degree less than  $d$ .

## 2.1 UC Framework

In this work, we use the *Universal Composability* (UC) framework [28] for security proofs. UC follows the simulation-based paradigm where the security of a protocol is defined with respect to an ideal world where a trusted party, the functionality  $\mathcal{F}$ , does the all of the computation. Informally, a protocol securely realizes  $\mathcal{F}$  in the real world if for any real world adversary there exist an equivalent ideal world adversary (the simulator). Equivalent meaning that any outside observer (the environment) cannot distinguish between the real protocol execution and the ideal execution. UC’s composition theorem ensures that one can safely compose protocols that have been proven UC-secure.

**Global Random Oracle.** More precisely, we use the generalized UC (GUC) framework [29] which allows to model global functionalities that are shared between different protocol instances. We consider a hybrid-model where parties have access to a (non-programmable) global random oracle  $\mathcal{G}_{\text{RO}}$  as introduced in [30]. We follow the simplified description from [25]. The  $\mathcal{G}_{\text{RO}}$  functionality can be queried by any party and the ideal adversary with two commands: QUERY and OBSERVE. The environment can query  $\mathcal{G}_{\text{RO}}$  by spawning additional dummy parties outside the context of the current protocol execution.  $\mathcal{G}_{\text{RO}}$  answers all new QUERY command by lazy sampling from the domain and stores them locally in a list  $\mathcal{Q}$ . A repeated query requires a simple lookup in  $\mathcal{Q}$ . Some QUERY queries are marked “illegitimate” and can be observed via OBSERVE command. Next we explain which query counts as an illegitimate one. Each party is associated with its party identifier  $\text{pid}$  and a session identifier  $\text{sid}$ . When a party queries  $\mathcal{G}_{\text{RO}}$  with the command (QUERY,  $x$ ), the query is parsed as  $(s, x')$  where  $s$  denotes the session identifier associated with the party. A query is marked as illegitimate if the  $\text{sid}$  field of the query differs from the  $\text{sid}$  associated to the party making the query. In other words, these are the queries made outside the context of the current session execution. We formally define the functionality  $\mathcal{G}_{\text{RO}}$  in Fig. 1.

*Remark 2.1.* In [30] the random oracle allows ideal functionalities to obtain the list of illegitimate queries. In order for the adversary to fetch those queries there needs to be a (dummy) functionality that forwards those queries. In [25] this is simplified by allowing the adversary to directly query the random oracle for illegitimate queries. Thus, functionalities no longer need to forward the illegitimate queries.

**Functionality 1:**  $\mathcal{G}_{\text{RO}}$

$\mathcal{G}_{\text{RO}}$  is parametrized by the output length  $\ell(1^\lambda)$ .

- **Query** Upon receiving a query (QUERY,  $x$ ), from some party  $\mathcal{P} = (\text{pid}, \text{sid})$  or from the adversary Sim do:
  - Look up  $v$  if there is a pair  $(x, v)$  for some  $v \in \{0, 1\}^{\ell(1^\lambda)}$  in the (initially empty) list  $\mathcal{Q}$  of past queries. Else, choose uniformly  $v \in \{0, 1\}^{\ell(1^\lambda)}$  and store the pair  $(x, v)$  in  $\mathcal{Q}$ .
  - 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  $s$ , that is denoted by  $\mathcal{Q}_{|s}$ .
  - Return  $v$  to  $\mathcal{P}$ .
- **Observe** Upon receiving a request (OBSERVE,  $\text{sid}$ ) from the adversary Sim, return the list  $\mathcal{Q}_{|\text{sid}}$  of illegitimate queries for SID  $\text{sid}$  to the adversary.

**Fig. 1.** Functionality for Global Random Oracle  $\mathcal{G}_{\text{RO}}$  [25]

Intuitively, these illegitimate queries are required for proving security of our protocols. The ideal adversary (or the simulator) works by observing  $\mathcal{G}_{\text{RO}}$  queries made by the corrupt party during the protocol execution. However, the environment can bypass this by querying  $\mathcal{G}_{\text{RO}}$  via additional dummy parties outside the current session. The simulator remains oblivious to these additional parties and thus fails in proving security. However, this behavior of the environment is accounted for in [25] by marking such queries as illegitimate and disclosing them to the simulator via OBSERVE command. Note that any  $\mathcal{G}_{\text{RO}}$  query for session id  $\text{sid}$  made by a party (or the simulator) participating in the session identified by  $\text{sid}$  will never be marked as illegitimate. Thus, any query made the simulator itself is not recorded by the functionality and hence cannot be observed by anyone. This is crucial for proving indistinguishability between the ideal and the real world and we elaborate in the proof of Theorem 3.1.

**Definition 2.2. (UC Security in the Global ROM [29,30]).** Let  $\mathcal{F}, \mathcal{F}'$  be  $m$ -party functionalities and  $\Pi$  be a protocol. We say that  $\Pi$  UC-realizes  $\mathcal{F}$  in the  $\mathcal{G}_{\text{RO}}, \mathcal{F}'$ -hybrid model if for any hybrid-model PPT adversary  $\mathcal{A}$ , there exists an ideal process PPT adversary Sim such that for every PPT environment  $\mathcal{Z}$ , it holds that:

$$\{\text{IDEAL}_{\mathcal{F}, \text{Sim}, \mathcal{Z}}^{\mathcal{G}_{\text{RO}}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x}, \lambda, z} \approx \{\text{REAL}_{\mathcal{F}', \Pi, \mathcal{A}, \mathcal{Z}}^{\mathcal{G}_{\text{RO}}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x}, \lambda, z}$$

where REAL is the outputs of the honest parties and the adversary  $\mathcal{A}$  after a real execution of protocol  $\Pi$  with input  $\mathbf{x} = (x_1, \dots, x_m)$  for parties  $P_1, \dots, P_m$  where each  $x_i \in \{0, 1\}^*$ ,  $z \in \{0, 1\}^*$  is the auxiliary input for  $\mathcal{A}$  and  $\lambda$  is the security parameter. IDEAL is the analogous distribution in an ideal execution

with a trusted party that computes  $\mathcal{F}$  for the parties and hands the output to the designated players.

## 2.2 Succinct Non Interactive Zero-Knowledge Proof

A *non-interactive proof system* for relation  $\mathcal{R}$ , denoted by  $\Pi_{\mathcal{R}}$ , consists a tuple of algorithms  $(\text{PGen}, \mathcal{O}_{\text{Setup}}, \mathcal{P}, \mathcal{V})$ .

- $\text{pp} \leftarrow \text{PGen}(1^\lambda)$ : Takes as input the security parameter  $\lambda$  and outputs public parameters  $\text{pp}$ . Once  $\text{PGen}$  is invoked we assume that all of the following algorithms take  $\text{pp}$  as an implicit input.
- $\text{out} \leftarrow \mathcal{O}_{\text{Setup}}(\text{in})$ : A stateful setup oracle that takes an input string  $\text{in}$  and outputs  $\text{out}$ .
- $\pi \leftarrow \mathcal{P}^{\mathcal{O}_{\text{Setup}}}(x, w)$ : Takes as input a statement  $x$  and witness  $w$ , and outputs a proof  $\pi$  if  $(x, w) \in \mathcal{R}$ .
- $b \leftarrow \mathcal{V}^{\mathcal{O}_{\text{Setup}}}(x, \pi)$ : Takes as input a statement  $x$  and proof  $\pi$ , and outputs a bit  $b$ , indicating “accept” or “reject”.

We introduce the setup oracle  $\mathcal{O}_{\text{Setup}}$  to the notation of NIZKs to capture the two typical setup assumptions in an abstract manner. That is, if a proof system is instantiated in the CRS model, then  $\mathcal{O}_{\text{Setup}}$  internally generates  $\text{crs}$  upon receiving a query with any input for the first time, and keeps outputting the same  $\text{crs}$  regardless of the input. When instantiating the RO model,  $\mathcal{O}_{\text{Setup}}$  is initialized with an empty query-response table and proceeds as follows. On receiving  $\text{in} \in \{0, 1\}^*$ , if  $\text{in}$  has never been queried, sample uniform  $\text{out} \in \{0, 1\}^{\ell(\lambda)}$ , store  $(\text{in}, \text{out})$  in the table, and return  $\text{out}$ . Otherwise, look up the table to find  $\text{out}$  associated with  $\text{in}$ , and return  $\text{out}$ .

We define three basic security properties for  $\Pi_{\mathcal{R}}$  in the stand-alone setting.

**Definition 2.3. (Completeness).**  $\Pi_{\mathcal{R}}$  satisfies *completeness* if for every  $(x, w) \in \mathcal{R}$ , it holds that

$$\Pr [b = 1 : \text{pp} \leftarrow \text{PGen}(1^\lambda); \pi \leftarrow \mathcal{P}^{\mathcal{O}_{\text{Setup}}}(x, w); b \leftarrow \mathcal{V}^{\mathcal{O}_{\text{Setup}}}(x, \pi)] = 1.$$

We define zero-knowledge by following the syntax of [38, 44]. A zero-knowledge simulator  $\mathcal{S}$  is defined as a stateful algorithm with initial state  $\text{st} = \text{pp}$  that operates in two modes. The first mode,  $(\text{out}, \text{st}') \leftarrow \mathcal{S}(1, \text{st}, \text{in})$  takes care of handling calls to the oracle  $\mathcal{O}_{\text{Setup}}$  on input  $\text{in}$ . The second mode,  $(\pi, \text{st}') \leftarrow \mathcal{S}(2, \text{st}, x)$  simulates a proof for the input statement  $x$ . For convenience we define three “wrapper” oracles. These oracles are stateful and share the internal state  $\text{st}$ , which initially contains an empty string.

- $\mathcal{S}_1(\text{in})$  to denote the oracle that returns the first output of  $\mathcal{S}(1, \text{st}, \text{in})$ ;
- $\mathcal{S}_2(x, w)$  that returns the first output of  $\mathcal{S}(2, \text{st}, x)$  if  $(x, w) \in \mathcal{R}$  and  $\perp$  otherwise;
- $\mathcal{S}'_2(x)$  that returns the first output of  $\mathcal{S}(2, \text{st}, x)$ .

**Definition 2.4. ((Unbounded) Zero-Knowledge).** Let  $\Pi_{\mathcal{R}} = (\text{PGen}, \mathcal{O}_{\text{Setup}}, \mathcal{P}, \mathcal{V})$  be a non-interactive proof system for relation  $\mathcal{R}$ .  $\Pi_{\mathcal{R}}$  is unbounded *non-interactive zero-knowledge* (NIZK), if there exists a PPT simulator  $\mathcal{S}$  with wrapper oracles  $\mathcal{S}_1$  and  $\mathcal{S}_2$  such that for all PPT adversaries  $\mathcal{A}$  it holds that

$$\left| \Pr \left[ b = 1 : \begin{array}{l} \text{pp} \leftarrow \text{PGen}(1^\lambda); \\ b \leftarrow \mathcal{A}^{\mathcal{O}_{\text{Setup}}, \mathcal{P}}(\text{pp}) \end{array} \right] - \Pr \left[ b = 1 : \begin{array}{l} \text{pp} \leftarrow \text{PGen}(1^\lambda); \\ b \leftarrow \mathcal{A}^{\mathcal{S}_1, \mathcal{S}_2}(\text{pp}) \end{array} \right] \right| < \text{negl}(\lambda).$$

Next, we define simulation extractability, which essentially guarantees that proofs are *non-malleable*. We stress that the present definition is weaker than what is necessary for realizing UC security, because the extractor algorithm here is *non-black-box*, i.e., it requires looking into the code of the adversary. The definition is an abstracted version of [53] and the schemes satisfying their definition clearly meet the version below by instantiating  $\mathcal{S}$  with trapdoor'd CRS generator in mode 1 and ZK simulator in mode 2.

**Definition 2.5. ((Non-black-box) Simulation Extractability).** Consider a non-interactive proof system  $\Pi_{\mathcal{R}} = (\text{PGen}, \mathcal{O}_{\text{Setup}}, \mathcal{P}, \mathcal{V})$  for relation  $\mathcal{R}$  with an NIZK simulator  $\mathcal{S}$ . Let  $(\mathcal{S}_1, \mathcal{S}'_2)$  be wrapper oracles for  $\mathcal{S}$  as defined above.  $\Pi_{\mathcal{R}}$  is non-black-box *simulation-extractable* (SIM-EXT) with respect to  $\mathcal{S}$ , if for any PPT adversary  $\mathcal{A}$ , there exists a PPT extractor  $\mathcal{E}_{\mathcal{A}}$  such that

$$\Pr \left[ \begin{array}{l} (x, \pi) \notin \mathcal{Q} \wedge (x, w) \notin \mathcal{R} \\ \wedge b = 1 \end{array} : \begin{array}{l} \text{pp} \leftarrow \text{PGen}(1^\lambda); (x, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_1, \mathcal{S}'_2}(\text{pp}); \\ b \leftarrow \mathcal{V}^{\mathcal{S}_1}(x, \pi); w \leftarrow \mathcal{E}_{\mathcal{A}}(x, \pi, \text{state}_{\mathcal{A}}, \text{st}) \end{array} \right] < \text{negl}(\lambda)$$

where  $\text{st}$  is the final state of the simulator  $\mathcal{S}$ ,  $\text{state}_{\mathcal{A}}$  is a string containing all inputs and outputs of  $\mathcal{A}$ , including random coins, and  $\mathcal{Q}$  is a set of statement-proof pairs  $(x, \pi)$  with  $x$  being a statement queried by  $\mathcal{A}$  to the proof simulation wrapper oracle  $\mathcal{S}'_2$ , and  $\pi$  being the corresponding simulated proof, respectively.

The ideal functionality  $\mathcal{F}_{\text{Setup}}$  that provides the setup and oracle for non-interactive proof system  $\Pi_{\mathcal{R}} = (\text{PGen}, \mathcal{O}_{\text{Setup}}, \mathcal{P}, \mathcal{V})$  is described in Fig. 2.

Our final goal is to compile  $\Pi_{\mathcal{R}}$  with the above basic security properties into a UC-secure NIZK protocol  $\Pi_{\text{UC-}\mathcal{R}}$ . The ideal functionality for Non-interactive Zero-Knowledge  $\mathcal{F}_{\text{NIZK}}$  is defined in Fig. 3. The functionality is taken from [55] with a minor difference being that  $\mathcal{F}_{\text{NIZK}}$  explicitly informs Sim of the associated session ID.

### 2.3 Succinct Polynomial Commitment Scheme

The following definition is adapted from the full version of [33]. The difference is that we omit the commitment key trimming algorithm as it is only necessary for concrete optimization.

**Functionality 2:  $\mathcal{F}_{\text{Setup}}$** 

$\mathcal{F}_{\text{Setup}}$  is parametrized by a security parameter  $\lambda$  and a degree bound  $D > 0$  and runs with parties  $P_1, \dots, P_N$  and an ideal process adversary  $\text{Sim}$ .

- **Parameters** Upon receiving input  $(\text{GENPARAMS}, \text{sid})$  from a party  $P_i$ , if no  $\text{pp}$  has been stored, run  $\text{pp} \leftarrow \text{PGen}(1^\lambda)$ , initialize oracle  $\mathcal{O}_{\text{Setup}}$  with  $\text{pp}$ , and store  $\text{pp}$ . Send  $(\text{PARAMS}, \text{sid}, \text{pp})$  to  $P_i$ .
- **Commitment Key** Upon receiving input  $(\text{GENKEY}, \text{sid})$  from a party  $P_i$ , if no  $\text{ck}$  has been stored, run  $\text{ck} \leftarrow \text{KGen}(1^\lambda, D)$  and store  $\text{ck}$ . Send  $(\text{COMKEY}, \text{sid}, \text{ck})$  to  $P_i$ .
- **Setup** Upon receiving input  $(\text{SETUP}, \text{sid}, \text{in})$  from a party  $P_i$ , ignore if  $\mathcal{O}_{\text{Setup}}$  has not been initialized with  $\text{pp}$ . Otherwise run  $\text{out} \leftarrow \mathcal{O}_{\text{Setup}}(\text{in})$  using the current state of  $\mathcal{O}_{\text{Setup}}$  and send  $(\text{SETUP}, \text{sid}, \text{out})$  to  $P_i$ .

**Fig. 2.**  $N$ -party functionality for setup  $\mathcal{F}_{\text{Setup}}$

**Definition 2.6. (Polynomial Commitment Scheme).** A polynomial commitment scheme over field  $\mathbb{F}$ , denoted by PCS, is a tuple of algorithms  $(\text{KGen}, \text{Com}, \text{Eval}, \text{Check})$ :

1.  $\text{ck} \leftarrow \text{KGen}(1^\lambda, D)$ : Takes as input the security parameter  $\lambda$  and the maximum degree bound  $D$  and generates commitment key  $\text{ck}$  as output.
2.  $c \leftarrow \text{Com}(\text{ck}, f, d; \rho_c)$ : Takes as input  $\text{ck}$ , the polynomial  $f \in \mathbb{F}_{<d}[X]$ , the degree bound  $d \leq D$ , randomness  $\rho_c$  and outputs a commitment  $c$ . In case the commitment scheme is deterministic  $\rho_c = \perp$ .
3.  $\pi \leftarrow \text{Eval}(\text{ck}, c, d, z, y, f; \rho_c)$ : Takes as input  $\text{ck}$ , the commitment  $c$ , degree bound  $d \leq D$ , evaluation point  $z \in \mathbb{F}$ , claimed polynomial evaluation  $y \in \mathbb{F}$ , the polynomial  $f$ , and outputs a non-interactive proof of evaluation  $\pi$ . The randomness  $\rho_c$  must equal the one previously used in  $\text{Com}$ .
4.  $b \leftarrow \text{Check}(\text{ck}, c, d, z, y, \pi)$ : Takes as input statement  $(\text{ck}, c, d, z, y)$  and the proof of evaluation  $\pi$  and outputs a bit  $b$ .

satisfying the following properties:

**Completeness.** For any integer  $1 \leq d \leq D$ , for all polynomials  $f \in \mathbb{F}_{<d}[X]$ , for all evaluation points  $z \in \mathbb{F}$ , and any randomness  $\rho_c$

$$\Pr \left[ \begin{array}{l} \text{ck} \leftarrow \text{KGen}(1^\lambda, D); c \leftarrow \text{Com}(\text{ck}, f, d; \rho_c); \\ b = 1 : y := f(z); \pi \leftarrow \text{Eval}(\text{ck}, c, d, z, y, f; \rho_c); \\ b \leftarrow \text{Check}(\text{ck}, c, d, z, y, \pi) \end{array} \right] = 1.$$

**Evaluation Binding.** For any integer  $1 \leq d \leq D$ , for all PPT adversaries  $\mathcal{A}$ ,

**Functionality 3:  $\mathcal{F}_{\text{NIZK}}$**

$\mathcal{F}_{\text{NIZK}}$  is parametrized by polynomial-time-decidable relation  $\mathcal{R} \in \{0, 1\}^* \times \{0, 1\}^*$ , and runs with parties  $P_1, \dots, P_N$  and an ideal process adversary  $\text{Sim}$ . It stores proof table  $\mathcal{Q}$  which is initially empty.

- **Proof** Upon receiving input  $(\text{PROVE}, \text{sid}, \text{ssid}, x, w)$  from a party  $P_i$ , ignore if  $(x, w) \notin \mathcal{R}$ . Otherwise, send  $(\text{PROVE}, \text{sid}, x)$  to  $\text{Sim}$ . Upon receiving  $(\text{PROOF}, \pi)$  from  $\text{Sim}$ , store  $(x, \pi)$  in  $\mathcal{Q}$  and send  $(\text{PROOF}, \text{sid}, \text{ssid}, \pi)$  to  $P_i$ .
- **Verification** Upon receiving input  $(\text{VERIFY}, \text{sid}, \text{ssid}, x, \pi)$  from a party  $P_i$ , if  $(x, \pi)$  is not stored in  $\mathcal{Q}$ , then send  $(\text{VERIFY}, \text{sid}, x, \pi)$  to  $\text{Sim}$ . Upon receiving  $(\text{WITNESS}, w)$  from  $\text{Sim}$ , if  $(x, w) \in \mathcal{R}$ , store  $(x, \pi)$  in  $\mathcal{Q}$ . Finally, return  $(\text{VERIFICATION}, \text{sid}, \text{ssid}, (x, \pi) \in? \mathcal{Q})$  to  $P_i$ .

**Fig. 3.**  $N$ -party functionality for non-interactive zero-knowledge  $\mathcal{F}_{\text{NIZK}}$

$$\Pr \left[ \begin{array}{l} y \neq y' \\ \wedge b = 1 \\ \wedge b' = 1 \end{array} : \begin{array}{l} \text{ck} \leftarrow \text{KGen}(1^\lambda, D); (c, d, z, y, y', \pi, \pi') \leftarrow \mathcal{A}(\text{ck}); \\ b \leftarrow \text{Check}(\text{ck}, c, d, z, y, \pi); \\ b' \leftarrow \text{Check}(\text{ck}, c, d, z, y', \pi') \end{array} \right] \leq \text{negl}(\lambda).$$

**Succinctness.** A **PCS** is said to be *succinct* if both the size of commitment  $c$  and evaluation proof  $\pi$  is of size  $\mathcal{O}_\lambda(1)$ .

In addition to standard properties above, we need a few more special properties for our compiler to work. In a later section we show that the widely used scheme of [57] indeed satisfy these.

**Definition 2.7. (Unique Proof).** For all PPT adversaries  $\mathcal{A}$ ,

$$\Pr \left[ \begin{array}{l} \pi \neq \pi' \\ \wedge b = 1 \\ \wedge b' = 1 \end{array} : \begin{array}{l} \text{ck} \leftarrow \text{KGen}(1^\lambda, D); \\ (c, d, z, y, \pi, \pi') \leftarrow \mathcal{A}(\text{ck}); \\ b \leftarrow \text{Check}(\text{ck}, c, d, z, y, \pi); \\ b' \leftarrow \text{Check}(\text{ck}, c, d, z, y, \pi') \end{array} \right] \leq \text{negl}(\lambda).$$

We define a polynomial encoding scheme, which takes a vector of field elements and outputs an appropriate randomized polynomial. An important property, sometimes referred to as *bounded independence* in the literature [33, §2.3]<sup>6</sup>, guarantees that a bounded number of evaluations do not leak any information about the original polynomial.

**Definition 2.8. (Polynomial Encoding Scheme).** A *polynomial encoding scheme*, denoted by **PES**, is a tuple of algorithms  $(\text{Enc}, \text{Dec})$  defined over an evaluation domain  $\mathcal{D}_{\text{Enc}}$  (which also determines the forbidden domain  $\mathcal{S}_{\text{Enc}} = \mathbb{F} \setminus \mathcal{D}_{\text{Enc}}$ ).

<sup>6</sup> This property is also known as  $k$ -knowledge bound in [13].



- $f \leftarrow \text{Enc}(\mathbf{w}, n, \ell; \boldsymbol{\rho})$ : Takes as inputs  $\mathbf{w} \in \mathbb{F}^n$ , dimension of the vector  $n > 0$ , evaluation bound  $\ell > 0$ , and randomness  $\boldsymbol{\rho} \in \mathbb{F}^\ell$ , and outputs a polynomial  $f \in \mathbb{F}_{<n+\ell}[X]$ .
- $\mathbf{w}' \leftarrow \text{Dec}(f, n, \ell)$ : Takes as inputs  $f \in \mathbb{F}_{<n+\ell}[X]$ ,  $n > 0$ , and  $\ell > 0$ , and deterministically outputs  $\mathbf{w}' \in \mathbb{F}^n$ .

We say PES is *correct* if  $\mathbf{w} = \text{Dec}(\text{Enc}(\mathbf{w}, n, \ell; \boldsymbol{\rho}), n, \ell)$  for any  $n > 0$ ,  $\ell > 0$ ,  $\mathbf{w} \in \mathbb{F}^n$ , and  $\boldsymbol{\rho} \in \mathbb{F}^\ell$ . PES satisfies *bounded independence* if for any  $n > 0$ ,  $\ell > 0$ , and  $\mathbf{w} \in \mathbb{F}^n$ , and for  $\boldsymbol{\rho}$  sampled uniformly from  $\mathbb{F}^\ell$ , any set of  $\ell$  evaluations of  $f \leftarrow \text{Enc}(\mathbf{w}, n, \ell; \boldsymbol{\rho})$  in  $\mathcal{D}_{\text{Enc}}$  are independently and uniformly distributed in  $\mathbb{F}$ .

In this work, we only consider polynomial encoding schemes where the size of the evaluation domain is exponential in the security parameter, i.e.  $|\mathcal{D}_{\text{Enc}}| \in O(2^\lambda)$ . Below we recall some candidate encoding schemes that are implicitly employed in many SNARK constructions.

- **Coefficient Encoding**  $\text{PES}_1 = (\text{Enc}_1, \text{Dec}_1)$ : PES can be instantiated using simple coefficient encoding as in [67]. Here  $\mathcal{D}_{\text{Enc}} = \mathbb{F} \setminus \{0\}$  and  $\text{Enc}_1$  outputs

$$f(X) = \sum_{i=1}^n w_i X^{i-1} + \sum_{i=1}^{\ell} \rho_i X^{n+i-1}$$

where  $\mathbf{w} = (w_i)_{i \in [n]}$  and  $\boldsymbol{\rho} = (\rho_i)_{i \in [\ell]}$ . The decoding algorithm  $\text{Dec}_1$  outputs the first  $n$  coefficients of  $f$ . It satisfies bounded independence because any set of  $\ell$  evaluations of  $f$  are independent of the encoded vector.

- **Lagrange Encoding**  $\text{PES}_2 = (\text{Enc}_2, \text{Dec}_2)$ : This encoding method has been used in e.g. [41, 33, 26]. Suppose a subset  $H \subset \mathbb{F}$  of cardinality  $n$  and an evaluation domain  $\mathcal{D}_{\text{Enc}} = \mathbb{F} \setminus (H \cup \{0\})$ . Assume that an input  $\mathbf{w} \in \mathbb{F}^n$  is indexed by  $H$ , i.e.,  $\mathbf{w} = (\mathbf{w}(a))_{a \in H}$ . Let  $L_{a,H} \in \mathbb{F}_{<n}[X]$  for  $a \in H$  be the Lagrange polynomials corresponding to  $H$  and  $Z_H(X) = \prod_{a \in H} (X - a)$  be a *vanishing polynomial of  $H$* . Then using  $\boldsymbol{\rho} = (\rho_i)_{i \in [\ell]}$  as randomness,  $\text{Enc}_2(\mathbf{w}, n, \ell; \boldsymbol{\rho})$  outputs

$$f(X) = \sum_{a \in H} \mathbf{w}(a) \cdot L_{a,H}(X) + \left( \sum_{i=1}^{\ell} \rho_i X^{i-1} \right) \cdot Z_H(X).$$

The decoding algorithm  $\text{Dec}_2$  outputs  $(f(a))_{a \in H}$ . On the one hand,  $\text{PES}_2$  satisfies correctness since  $f$  agrees with  $\mathbf{w}$  over the forbidden domain  $\mathcal{S}_{\text{Enc}} = H$ . On the other hand, up to  $\ell$  evaluations of  $f$  in  $\mathcal{D}_{\text{Enc}}$  reveal nothing about the encoded vector  $\mathbf{w}$ . Typically, the evaluation bound  $\ell$  should be set strictly larger than the number of evaluation proofs the prover explicitly reveals, because a commitment to the polynomial itself may leak information about one evaluation (as in the KZG scheme). It turns out that this property helps us show the hiding property below once combined with a suitable polynomial commitment scheme.

**Evaluation Hiding.** We now define *evaluation hiding*. Note that this is a stronger property than the usual hiding definition (such as the ones in [57, 33]):

essentially, evaluation hiding guarantees that the joint distribution of commitment, evaluation proof, and polynomial evaluations leaks nothing about the committed polynomial, whereas the usual PCS hiding property does allow evaluations to be associated with the committed polynomials. Clearly, if Enc is deterministic PCS can never be evaluation hiding. This is why the definition only makes sense with respect to a specific encoding scheme. Recent IOP-based SNARKs such as [41, 67, 33, 26] in fact exploit this property (albeit without formal definition tailored to PCS) to hide evaluations of a polynomial encoding secret witness and thus to retain perfect zero knowledge. The definition is parameterized by a function  $\phi : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$  calculating the expansion factor for encoding randomness: given the number of evaluated points  $\ell' > 0$ , it determines  $\ell > \ell'$  the total number of random field elements necessary for hiding the committed polynomial *even after outputting a commitment,  $\ell'$  evaluation proofs, and  $\ell'$  evaluations*.

**Definition 2.9. ( $\phi$ -Evaluation Hiding).** Let  $\text{PCS} = (\text{KGen}, \text{Com}, \text{Eval}, \text{Check}) = (\text{KGen}, \text{Com}, \text{Eval}, \text{Check})$  be a polynomial commitment scheme and  $\text{PES} = (\text{Enc}, \text{Dec})$  be a polynomial encoding scheme. We say  $\text{PCS}$  is  $\phi$ -evaluation hiding with respect to PES if for all PPT adversaries  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ ,

$$\Pr \left[ \begin{array}{l} \text{ck} \leftarrow \text{KGen}(1^\lambda, D); (\mathbf{w}, \mathbf{z}) \leftarrow \mathcal{A}_1(\text{ck}); \\ n := |\mathbf{w}|; \ell := \phi(|\mathbf{z}|); d := n + \ell; \\ \rho_w \leftarrow_{\$} \mathbb{F}^\ell; b \leftarrow_{\$} \{0, 1\}; \\ f \leftarrow \text{Enc}(b \cdot \mathbf{w}, n, \ell; \rho_w); \\ c \leftarrow \text{Com}(\text{ck}, f, d; \rho_c); \\ \mathbf{y} := f(\mathbf{z}); \\ \pi \leftarrow \text{Eval}(\text{ck}, c, d, \mathbf{z}, \mathbf{y}, f; \rho_c); \\ b' \leftarrow \mathcal{A}_2(c, \mathbf{y}, \pi) \end{array} : b = b' \wedge \mathbf{z} \in \mathcal{D}_{\text{Enc}}^{|\mathbf{z}|} \right] \leq \frac{1}{2} + \text{negl}(\lambda)$$

where  $\mathcal{A}_1, \mathcal{A}_2$  share the internal states,  $\mathbf{y} := f(\mathbf{z})$  denotes setting  $y_i := f(z_i)$  for all  $i \in [|\mathbf{z}|]$ , and  $\pi \leftarrow \text{Eval}(\text{ck}, c, d, \mathbf{z}, \mathbf{y}, f; \rho_c)$  denotes setting  $\pi_i \leftarrow \text{Eval}(\text{ck}, c, d, z_i, y_i, f; \rho_c)$  for all  $i \in [|\mathbf{z}|]$ .

**Non-Extrapolation.** We define a new property related to  $\phi$ -evaluation hiding of a PCS scheme with respect to a PES scheme. We require that, given a polynomial commitment and  $\ell' > 0$  evaluations and proofs for an encoding of all-zero vector, no adversary can compute a valid proof for a new evaluation point. In other words, it is hard for an adversary to *extrapolate* a new evaluation given  $\ell'$  evaluations *even when* the polynomial is fixed to be the encoding of all-zero vector. Non-extrapolation naturally follows from evaluation hiding and binding for many PCS plus PES schemes for the right choice of  $\phi$ . We show this explicitly for the KZG polynomial commitment scheme in Section 4.

**Definition 2.10. ( $\phi$ -Non-Extrapolation).** Let  $\text{PCS} = (\text{KGen}, \text{Com}, \text{Eval}, \text{Check}) = (\text{KGen}, \text{Com}, \text{Eval}, \text{Check})$  be a polynomial commitment scheme and

$PES = (\text{Enc}, \text{Dec})$  be a polynomial encoding scheme. We say  $PCS$  supports  $\phi$ -*non-extrapolation with respect to*  $PES$  if for all PPT adversaries  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ , and

$$\Pr \left[ \begin{array}{l} \text{ck} \leftarrow \text{KGen}(1^\lambda, D); (n, \mathbf{z}) \leftarrow \mathcal{A}_1(\text{ck}); \\ \ell := \phi(|\mathbf{z}|); d := n + \ell; \\ \rho_w \leftarrow_{\mathfrak{s}} \mathbb{F}^\ell; \\ f \leftarrow \text{Enc}(0^n, n, \ell; \rho_w); \\ c \leftarrow \text{Com}(\text{ck}, f, d; \rho_c); \\ \mathbf{y} := f(\mathbf{z}); \\ \pi \leftarrow \text{Eval}(\text{ck}, c, d, \mathbf{z}, \mathbf{y}, f; \rho_c); \\ (z^*, y^*, \pi^*) \leftarrow \mathcal{A}_2(c, \mathbf{y}, \pi); \\ v \leftarrow \text{Check}(\text{ck}, c, d, z^*, y^*, \pi^*) \end{array} \right] \leq \text{negl}(\lambda)$$

$$\Pr \left[ \begin{array}{l} v = 1 \wedge \mathbf{z} \in \mathcal{D}_{\text{Enc}}^{|\mathbf{z}|} : \\ \wedge z^* \in \mathcal{D}_{\text{Enc}} \wedge z^* \notin \mathbf{z} \end{array} \right]$$

where  $\mathcal{A}_1$  and  $\mathcal{A}_2$  share the internal states,  $\mathbf{y} := f(\mathbf{z})$ ,  $\pi$  are as before.

### 3 Succinctness-Preserving UC NIZK Compiler

In this section, we describe a generic, succinctness-preserving compiler that takes as inputs: (1) a **SIM-EXT NIZK** proof system  $\Pi_{\mathcal{R}} = (\text{PGen}, \mathcal{O}_{\text{Setup}}, \mathcal{P}, \mathcal{V})$  for the *arithmetic circuit satisfiability relation*  $\mathcal{R} = \{(\mathcal{C}, w) : \mathcal{C}(w) = 1\}$ , and (2) a  $PCS = (\text{KGen}, \text{Com}, \text{Eval}, \text{Check})$  with suitable properties. The resulting protocol, denoted by  $\Pi_{\text{UC-}\mathcal{R}}$ , UC-realizes  $\mathcal{F}_{\text{NIZK}}$  in the  $(\mathcal{G}_{\text{RO}}, \mathcal{F}_{\text{Setup}})$ -hybrid model, where  $\mathcal{F}_{\text{Setup}}$  is described in Fig. 2.

**Theorem 3.1.** *Let  $\Pi_{\mathcal{R}}$  be a **SIM-EXT NIZK** proof system, for the arithmetic circuit satisfiability relation  $\mathcal{R}$ , with  $\mathcal{O}_\lambda(1)$  size proofs. Let  $PCS$  be a polynomial commitment scheme with  $\mathcal{O}_\lambda(1)$  size commitments and evaluation proofs, evaluation binding, unique proofs,  $\phi$ -evaluation hiding, and  $\phi$ -non-extrapolation with respect to the encoding scheme  $PES = (\text{Enc}, \text{Dec})$ . Then,  $\Pi_{\text{UC-}\mathcal{R}}$  described in Fig. 4 UC-realizes  $\mathcal{F}_{\text{NIZK}}$  in the  $(\mathcal{G}_{\text{RO}}, \mathcal{F}_{\text{Setup}})$ -hybrid model for relation  $\mathcal{R}$  and has proofs of size  $\mathcal{O}_\lambda(1)$ .*

*Proof.* We prove the following properties.

**Completeness.** For a given commitment  $c$  and circuit  $\mathcal{C}'$ , an honest prover fails to generate a valid proof if, after trying at most  $T$  distinct evaluation points  $z_i$ 's  $\in \mathcal{D}_{\text{Enc}}$ , it fails to find any preimage such that it hashes to  $0^b$ . As we will see,  $T$  is required to be only polynomially big in  $\lambda$  and so the prover is guaranteed to stop in polynomial time. For each iteration  $i$ , after fixing  $c, \mathcal{C}', z_i$ , the values  $y_i = f(z_i)$  and  $\pi_i$  are derived uniquely. Thus, the honest prover fails in this iteration only if for all the  $T$  number of evaluation points  $\mathcal{G}_{\text{RO}}(\text{QUERY}, (\text{sid}, (\mathcal{C}', c, z_i, y_i, \pi_i, i))) \neq 0^b$ . The prover fails overall if it fails in at least one of the iterations. Let the event of failing in iteration  $i$  be denoted by  $\text{fail}_i$ . For  $T = (\lambda + \log(r)) \cdot 2^b$ , the probability of the honest prover failing can be bounded as below.

**Protocol 1:**  $\Pi_{\text{UC-}\mathcal{R}}$ 

The protocol  $\Pi_{\text{UC-}\mathcal{R}}$  is parameterized by: security parameter  $\lambda$ , finite field  $\mathbb{F}$ , evaluation domain  $\mathcal{D}_{\text{Enc}}$  for **PES**, evaluation hiding expansion factor  $\phi: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ , number of parallel repetitions  $r = r(\lambda) > 0$ , proof-of-work parameter  $b(\lambda) > 0$ , bound  $T(\lambda) > 0$ , and maximum degree bound  $D > 0$  for **PCS**.

- **Proof** Upon receiving input  $(\text{PROVE, sid, ssid, } \mathcal{C}, w)$ , ignore if  $\mathcal{C}(w) \neq 1$ . Otherwise,  $\mathcal{P}_i$  does:
  1. Send  $(\text{GENPARAMS, sid})$  to  $\mathcal{F}_{\text{Setup}}$  and wait for answer  $(\text{PARAMS, sid, pp})$ .
  2. Send  $(\text{GENKEY, sid})$  to  $\mathcal{F}_{\text{Setup}}$  and wait for answer  $(\text{COMKEY, sid, ck})$ .
  3. Parse  $w = \mathbf{w} \in \mathbb{F}^n$ . Let  $\ell := \phi(r)$  and  $d := n + \ell$ . If  $d > D$ , abort by outputting  $(\text{PROOF, sid, ssid, } \perp)$ .
  4. Generate a polynomial encoding of the witness vector:  $f \leftarrow \text{Enc}(\mathbf{w}, n, \ell; \rho_w)$ , where  $\rho_w \leftarrow \mathbb{F}^\ell$ .
  5. Generate a commitment to the polynomial encoding:  $c \leftarrow \text{Com}(\text{ck}, f, d; \rho_c)$ , where the randomness  $\rho_c$  is sampled uniformly from the domain specified in **PCS**.
  6. Define the circuit  $\mathcal{C}'$  such that it outputs 1 on input  $w' = (\mathbf{w}, \rho_w, \rho_c)$  if and only if the following conditions are met:

$$\mathcal{C}(\mathbf{w}) = 1 \wedge c = \text{Com}(\text{ck}, \text{Enc}(\mathbf{w}, n, \ell; \rho_w), d; \rho_c)$$

- 7. Run  $\Pi_{\mathcal{R}}.\mathcal{P}$  on input  $\text{pp}, \mathcal{C}'$ , and  $w'$  to obtain a proof  $\pi'$ . Whenever  $\mathcal{P}$  makes a call to  $\mathcal{O}_{\text{Setup}}$  with input  $\text{in}$ , send  $(\text{SETUP, sid, in})$  to receive a response  $(\text{SETUP, sid, out})$  and forward out to  $\mathcal{P}$ .
- 8. Initialize empty sets  $\mathbf{z}, \mathbf{y}$ , and  $\pi_{\text{PCS}}$ .
- 9. For each iteration  $i \in [r]$  do:
  - (a) Initialize counter  $\text{ctr} := 0$  and an empty set of used evaluation points  $\mathcal{D}_i$ .
  - (b) If  $\text{ctr} = T$ , abort by outputting  $(\text{PROOF, sid, ssid, runout\_eval})$ .
  - (c) Sample an evaluation point:  $z_i \leftarrow \mathcal{D}_{\text{Enc}} \setminus \mathcal{D}_i$ . Update  $\text{ctr} := \text{ctr} + 1$ . Update  $\mathcal{D}_i := \mathcal{D}_i \cup \{z_i\}$ .
  - (d) Compute  $y_i = f(z_i)$  and evaluation proof  $\pi_i \leftarrow \text{Eval}(\text{ck}, c, d, z_i, y_i, f; \rho_c)$ .
  - (e) Send  $(\text{QUERY, (sid, } (\mathcal{C}', c, z_i, y_i, \pi_i, i)))$  to  $\mathcal{G}_{\text{RO}}$ . Upon receiving  $v$  from  $\mathcal{G}_{\text{RO}}$ , if the first  $b$  bits of  $v$  are not  $0^b$ , go to step (b). Otherwise, store  $z_i, y_i$ , and  $\pi_i$  in  $\mathbf{z}, \mathbf{y}$ , and  $\pi_{\text{PCS}}$ , respectively.
- 10. Output  $(\text{PROOF, sid, ssid, } \varpi)$ , where  $\varpi := (\pi', c, \mathbf{z}, \mathbf{y}, \pi_{\text{PCS}})$ .
- **Verification** Upon receiving input  $(\text{VERIFY, sid, ssid, } \mathcal{C}, \varpi)$ ,  $\mathcal{P}_i$  does:
  1. Send  $(\text{GENPARAMS, sid})$  to  $\mathcal{F}_{\text{Setup}}$  and wait for answer  $(\text{PARAMS, sid, pp})$ .
  2. Send  $(\text{GENKEY, sid})$  to  $\mathcal{F}_{\text{Setup}}$  and wait for answer  $(\text{COMKEY, sid, ck})$ .
  3. Parse  $\varpi = (\pi', c, \mathbf{z}, \mathbf{y}, \pi_{\text{PCS}})$ . Derive the witness size  $n$  from the description of  $\mathcal{C}$ . Compute  $\ell$  and  $d$  as **Proof** would and if  $d > D$  abort by outputting  $(\text{VERIFICATION, sid, ssid, } 0)$ .
  4. Define the circuit  $\mathcal{C}'$  as **Proof** would.
  5. Parse  $\mathbf{z} = (z_i)_{i \in [r]}$ ,  $\mathbf{y} = (y_i)_{i \in [r]}$ , and  $\pi_{\text{PCS}} = (\pi_i)_{i \in [r]}$ .
  6. Output  $(\text{VERIFICATION, sid, ssid, } 1)$  if all of the following checks pass, otherwise output  $(\text{VERIFICATION, sid, ssid, } 0)$ :
    - (a)  $\Pi_{\mathcal{R}}.\mathcal{V}$  on input  $\text{pp}, \mathcal{C}'$ , and  $\pi'$  outputs 1. Calls to  $\mathcal{O}_{\text{Setup}}$  by  $\mathcal{V}$  are handled similar to the above.
    - (b) For all  $i \in [r]$ :  $1 = \text{Check}(\text{ck}, c, d, z_i, y_i, \pi_i)$ .
    - (c) For all  $i \in [r]$ : send  $(\text{QUERY, (sid, } (\mathcal{C}', c, z_i, y_i, \pi_i, i)))$  to  $\mathcal{G}_{\text{RO}}$ , and the first  $b$  bits of the return value  $v_i$  are  $0^b$ .

**Fig. 4.** Protocol for UC-secure non-interactive proof in the  $(\mathcal{G}_{\text{RO}}, \mathcal{F}_{\text{Setup}})$ -hybrid model.  $\Pi_{\text{UC-}\mathcal{R}}$  internally runs  $\Pi_{\mathcal{R}}$ , **PCS**, and **PES**.

$$\Pr[\text{fail}] \leq \sum_{i=1}^r \Pr[\text{fail}_i] = r \cdot \left(1 - \frac{1}{2^b}\right)^T \approx e^{\log(r)} \cdot \frac{1}{e^{(\lambda + \log(r))}} \leq 2^{-\lambda}$$

Thus, an honest prover manages to find a preimage of  $0^b$  in polynomial time except with probability  $2^{-\lambda}$ . We also remark that the completeness error increases only additively even if the underlying proof system  $\Pi_{\mathcal{R}}$  is statistically complete.<sup>7</sup> We defer the analysis in this case to the full version.

**Simulation.** We begin by sketching the overall simulation strategy. First, consider simulation for an uncorrupted prover. We simulate the behaviors of  $\mathcal{F}_{\text{Setup}}$  and  $\pi'$  component of real-world proofs produced by honest provers using the underlying NIZK simulator  $\Pi_{\mathcal{R}}.\mathcal{S}$ . After the first  $r$  queries to  $\mathcal{G}_{\text{RO}}$  are programmed to be  $0^b$ , commitments to witness-encoding polynomials are replaced with simulated commitments to randomized polynomials encoding a dummy witness (i.e., 0-vector). This transition is justified by the evaluation hiding property (Definition 2.9). Then we stop programming the  $\mathcal{G}_{\text{RO}}$  responses in the next hybrid. At this stage, simulation of uncorrupted provers is essentially done.

Next, we describe simulation for an uncorrupted verifier. The requirement here is to extract a witness from whatever  $(\tilde{\mathcal{C}}, \tilde{\omega} = (\tilde{\pi}', \tilde{c}, \tilde{\mathbf{z}}, \tilde{\mathbf{y}}, \tilde{\pi}_{\text{PCS}}))$  submitted by uncorrupted verifiers unless they have been created during the simulation of uncorrupted provers. Here, we first rule out the case where at least one of  $(\tilde{\mathbf{z}}, \tilde{\mathbf{y}}, \tilde{\pi}_{\text{PCS}})$  differs from previously simulated  $(\mathbf{z}, \mathbf{y}, \pi_{\text{PCS}})$  for the same statement  $\tilde{\mathcal{C}}$  and  $\tilde{c}$ . This can be shown by constructing a reduction to evaluation binding, evaluation hiding, or unique proof. Finally, the extraction algorithm interpolates the witness-encoding polynomial  $f$  for  $\tilde{c}$  by observing  $\mathcal{G}_{\text{RO}}$  queries and decodes  $f$  to a candidate witness  $w = \mathbf{w} \in \mathbb{F}^n$ . The analysis concludes by bounding the probability that extracted  $w$  is invalid as follows. We run a non-black-box SIM-EXT extractor  $\mathcal{E}_{\mathcal{Z}}$  of the underlying proof system against successful  $\mathcal{Z}$  on statement the extended circuit  $\tilde{\mathcal{C}}'$ , and proof  $\tilde{\pi}'$  to obtain another candidate witness  $w' = (\mathbf{w}^*, \rho_w, \rho_c)$ . This fails in the case that  $\tilde{\pi}'$  is a previously simulated proof. However, we rule this case out by relying on non-extrapolation property. Given this, the event  $\tilde{\mathcal{C}}'(w') = 0$  happens only with negligible probability thanks to the simulation-extractability property. Hence, assuming  $\tilde{\mathcal{C}}'(w') = 1$ , it also holds that  $\tilde{\mathcal{C}}(\mathbf{w}^*) = 1$  by the definition of  $\tilde{\mathcal{C}}'$ . Then we show that  $\mathbf{w} = \mathbf{w}^*$ . Otherwise, one can construct another witness-encoding polynomial  $f^* \neq f$  that “explains” the same commitment  $\tilde{c}$ , breaking evaluation binding. With this we conclude that the extracted witness  $\mathbf{w}$  is a valid witness as  $\mathbf{w} = \mathbf{w}^*$  and  $\tilde{\mathcal{C}}(\mathbf{w}^*) = 1$ .

The above proof sketch describes simulation strategy for a single prover and verifier. In the formal proof, this is extended to incorporate multiple uncorrupted provers and verifiers in a session.

Let us turn to formal proof. Complete simulation algorithm is given in Fig. 5. The environment  $\mathcal{Z}$  starts a session by initializing a certain number of parties and adversary  $\mathcal{A}$ . In a particular session  $\text{sid}$ , the environment  $\mathcal{Z}$  instructs the parties with two commands: PROVE and VERIFY. The real world behavior is as follows.

<sup>7</sup> We thank an anonymous reviewer for bringing this observation to our attention.

An honest party  $P_i$  on input  $(\text{PROVE}, \text{sid}, \text{ssid}, \mathcal{C}, w)$  from  $\mathcal{Z}$  executes the honest prover's algorithm in  $\Pi_{\text{UC-R}}$  to generate the proof. And on receiving  $(\text{VERIFY}, \text{sid}, \text{ssid}, \mathcal{C}, \varpi)$ , it verifies by running the honest verifier's algorithm. In the ideal world, the honest parties forward their inputs to the functionality  $\mathcal{F}_{\text{NIZK}}$ . The corrupt parties' behavior is controlled by  $\mathcal{A}$  in both the worlds. Within a session  $\text{sid}$ , we assume that  $\mathcal{Z}$  issues  $s_1$  queries of the type  $(\text{PROVE}, \text{sid}, \text{ssid}, \mathcal{C}, w)$  meant for an honest party, and  $s_2$  of the type  $(\text{VERIFY}, \text{sid}, \text{ssid}, \mathcal{C}, \varpi)$  for either honest or corrupt party. Let  $s = s_1 + s_2$ . Proofs for indistinguishability of hybrids are deferred to the full version [43].

- $\text{Hyb}_0$  : This is the real world.
- $\text{Hyb}_1$ : Replace all the honest parties with a single party  $\mathcal{B}$ . This party is responsible for simulating the view of the adversary and the environment for the rest of the protocol. In particular,  $\mathcal{B}$  acts on behalf of the honest parties and does exactly what an honest party would do in the real world. In addition, it intercepts the  $\mathcal{G}_{\text{RO}}$  queries made by any corrupt party  $P_i$  within the session, forwards it the  $\mathcal{G}_{\text{RO}}$  and relays the response back to  $P_i$ . Similarly, it intercepts all  $\mathcal{F}_{\text{Setup}}$  queries made by  $P_i$  in the session and relays it back and forth between  $\mathcal{F}_{\text{Setup}}$  and  $P_i$ .
- $\text{Hyb}_2$ : Instead of forwarding  $P_i$ 's calls to  $\mathcal{F}_{\text{Setup}}$  functionality,  $\mathcal{B}$  answers them by executing steps in **Simulation of  $\mathcal{F}_{\text{Setup}}$**  in **Sim**. The rest of the execution remains the same as before, i.e., the  $\mathcal{B}$  executes on behalf of the honest parties by executing the honest algorithm.
- For  $j \in [s_1]$ ,  $\text{Hyb}_{2+j}$ : For the  $j$ -th **PROVE** command with input  $(\mathcal{C}, w)$  for an honest party  $P_i$  from  $\mathcal{Z}$ , replace honest prover's algorithm with Step 1-7. in **Simulation of uncorrupted prover** (in **Sim**) for input  $\mathcal{C}$ .
- For  $j \in [s_2]$ ,  $\text{Hyb}_{2+s_1+j}$ : For the  $j$ -th **VERIFY** command with input  $(\mathcal{C}, \varpi)$  for an honest party  $P_i$  from  $\mathcal{Z}$ , replace honest verifier's algorithm with Step 1-12 in **Simulation of uncorrupted verifier** (in **Sim**). We assume that all the **VERIFY** commands are made only by the honest parties. This is without loss of generality as any query that a corrupt party wants to make can instead be routed through an honest party via the environment.
- $\text{Hyb}_{3+s}$ : This is the ideal world execution. Replace  $\mathcal{B}$  with **Sim**, where the steps in **Sim** are executed for each  $(\text{PROVE}, \text{sid}, \text{ssid}, \mathcal{C}, w)$ , and  $(\text{VERIFY}, \text{sid}, \text{ssid}, \mathcal{C}, \varpi)$  command (as explained in the above hybrids), and sends corresponding  $(\text{PROOF}, \text{sid}, \text{ssid}, P_i, \varpi)$  and  $(\text{WITNESS}, \text{sid}, \text{ssid}, P_i, w)$  to  $\mathcal{F}_{\text{NIZK}}$ .

**Succinctness.** From completeness and simulation analysis we obtain the following constraints for parameters  $r, b, T$ :  $T = (\lambda + \log(r)) \cdot 2^b$  and  $\lambda = r(b - \log(d))$ . Consider the simple parameter choice  $r = \lambda$ . This gives,  $b = \log(d) + 1$  and  $T = 2d(\lambda + \log(\lambda))$ . More generally, the parameter choices,  $r = \mathcal{O}(\lambda / \log(\lambda)) = \mathcal{O}_\lambda(1)$ ,  $b = \mathcal{O}(\log(d) + \log(\lambda)) = \mathcal{O}_\lambda(\log(d))$ , and  $T = \mathcal{O}((\lambda + \log(\lambda / \log(\lambda)))\lambda d) = \mathcal{O}_\lambda(d)$  satisfies the conditions.

Assume that **PCS** produces constant size  $(\mathcal{O}_\lambda(1))$  commitments and evaluation proofs, and  $\Pi_{\mathcal{R}}$  produces  $\mathcal{O}_\lambda(1)$  size proofs. Later in Sect. 4 we discuss candidate

schemes satisfying these constraints. The total size of the proof  $\varpi$  is one commitment  $c$  of size  $\mathcal{O}_\lambda(1)$ , vectors  $\mathbf{z}, \mathbf{y}$  consisting of  $r$  field elements,  $r$  evaluation proofs  $\pi_i$  of size  $\mathcal{O}_\lambda(1)$ , and one NIZK proof  $\pi'$  for statement  $\mathcal{C}'$ . Recall that  $\mathcal{C}'$  is composed of  $\mathcal{C}$  and the circuit that describes the Com and Enc operations. Thus,  $\mathcal{C}'$  is only  $\mathcal{O}(\text{poly}(\lambda, n))$  bigger than  $\mathcal{C}$ , where  $n$  is the witness size. Since  $\Pi_{\mathcal{R}}$  produces constant size proofs, proof for  $\mathcal{C}'$  is also of size  $\mathcal{O}_\lambda(1)$ . Finally, since,  $r = \mathcal{O}_\lambda(1)$ , the size of  $\varpi$  remains  $\mathcal{O}_\lambda(1)$ .

*Remark 3.2.* Here,  $r$  is independent of the degree of the polynomial. The proof size only grows with the number of repetitions and thus remains independent of the degree, assuming constant size PCS and NIZK  $\Pi_{\mathcal{R}}$  proofs. However, the prover's computational effort increases with the increase in degree  $d$ .

## 4 Instantiating Our Compiler

In this section, we discuss a few candidates for PCS, PES and NIZK schemes for instantiating our compiler.

### 4.1 A Candidate PCS and PES Scheme

We show that using KZG commitments [57] as the PCS scheme along with Coefficient (PES<sub>1</sub>) or Lagrange (PES<sub>2</sub>) encoding scheme (§ 2.3) satisfies all necessary conditions required to instantiate our compiler, i.e., it is succinct, has evaluation binding, has unique proofs, is evaluation hiding, and has non-extrapolation.

We describe the polynomial commitment scheme  $\text{PCS}_{\text{KZG}} = (\text{KGen}, \text{Com}, \text{Eval}, \text{Check})$ . The formalization below follows the deterministic scheme of [33, §C.2] supporting multiple degree bounds up to the maximum degree  $D$ . Note that if  $d = D$ , one can skip computing/checking  $\hat{c}, \hat{\pi}$ , and  $\hat{y}$ .

- $\text{KGen}(1^\lambda, D)$ : Generate the parameters of a bilinear group  $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, e)$  where  $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T| = q$  is prime,  $\langle g \rangle = \mathbb{G}_1$ ,  $\langle h \rangle = \mathbb{G}_2$ , and  $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$  is an efficiently computable, non-degenerate bilinear map. The group order  $q$  also determines  $\mathbb{F} := \mathbb{F}_q$  and a set of supported polynomials  $\mathbb{F}_{<D}[X]$ . Sample  $\alpha \in \mathbb{F}$  uniformly, and compute  $\sigma = (g, g^\alpha, \dots, g^{\alpha^{D-1}}, h, h^\alpha)$ . Output  $\text{ck} = (\mathcal{G}, \sigma)$ .
- $\text{Com}(\text{ck}, f, d)$ : On input  $\text{ck}$ , a polynomial  $f \in \mathbb{F}_{<d}[X]$ , and a degree bound  $d \leq D$ , compute a shifted polynomial  $\hat{f} = X^{D-d} \cdot f$ , and generate a commitment as  $\mathbf{c} = (g^{f(\alpha)}, g^{\hat{f}(\alpha)})$  and output  $\mathbf{c}$ .
- $\text{Eval}(\text{ck}, c, d, z, f(z), f)$ : Compute  $\omega(X) = (f(X) - f(z))/(X - z)$  and  $\hat{\omega}(X) = (\hat{f}(X) - \hat{f}(z))/(X - z)$  where  $\hat{f}$  is computed as above. Output  $\boldsymbol{\pi} = (g^{\omega(\alpha)}, g^{\hat{\omega}(\alpha)}, \hat{f}(z))$ .
- $\text{Check}(\text{ck}, \mathbf{c}, d, z, y, \boldsymbol{\pi})$ : Parse  $\mathbf{c} = (c, \hat{c})$  and  $\boldsymbol{\pi} = (\pi, \hat{\pi}, \hat{y})$ . Accept if and only if  $e(c/g^y, h) = e(\pi, h^\alpha/h^z)$ ,  $e(\hat{c}/g^{\hat{y}}, h) = e(\hat{\pi}, h^\alpha/h^z)$ , and  $\hat{y} = z^{D-d} \cdot y$ .

The security of  $\text{PCS}_{\text{KZG}}$  relies on the SDH assumption [18].

**Simulator: Sim**

Sim is parameterized by  $\lambda, \mathbb{F}, \mathcal{D}_{\text{Enc}}, \phi, T, r, b, D$  and has access to the global functionality  $\mathcal{G}_{\text{RO}}$  as  $\Pi_{\text{UC-R}}$  does. It simulates real prover's proof for arbitrary  $(C, w) \in \mathcal{R}$ , extracts a witness from a valid proof  $(C, \varpi)$  chosen by the environment (as long as it hasn't been recorded by  $\mathcal{F}_{\text{NIZK}}$ ), and simulates the local setup functionality  $\mathcal{F}_{\text{Setup}}$ . It internally keeps track of the state information st of the underlying NIZK simulator  $\Pi_{\mathcal{R}.S}$ , which is initially set to  $\varepsilon$ .

- **Initialization** follows [50]: We use the notation  $\tilde{P}_i$  for a dummy party in the ideal process, which simply forwards inputs and outputs between the environment  $\mathcal{Z}$  and the ideal functionality  $\mathcal{F}_{\text{NIZK}}$ , and  $P_i$  for a simulated party. Sim starts by invoking a copy of a PPT adversary  $\mathcal{A}$ . It will run a simulated interaction of  $\mathcal{A}$ , the parties, and the environment. In particular, whenever  $\mathcal{A}$  communicates with  $\mathcal{Z}$ , Sim just passes this information along. And whenever  $\mathcal{A}$  corrupts a party  $P_i$ , Sim corrupts the corresponding dummy party  $\tilde{P}_i$ .
- **Simulation of  $\mathcal{F}_{\text{Setup}}$** 
  - **Parameters** Upon receiving input  $(\text{GENPARAMS}, \text{sid})$  from a party  $P_i$ , if no  $\text{pp}$  has been stored, run  $\text{pp} \leftarrow \text{PGen}(1^\lambda)$ , let  $\text{st} := \text{pp}$ , and store  $\text{pp}$ . Send  $(\text{PARAMS}, \text{sid}, \text{pp})$  to  $P_i$ .
  - **Commitment Key** This is identical to  $\mathcal{F}_{\text{Setup}}$ .
  - **Setup** Upon receiving input  $(\text{SETUP}, \text{sid}, \text{in})$  from a party  $P_i$ , ignore if  $\text{st}$  has never been initialized with  $\text{pp}$ . Otherwise run  $(\text{out}, \text{st}) \leftarrow \Pi_{\mathcal{R}.S}(1, \text{st}, \text{in})$  using the current state and send  $(\text{SETUP}, \text{sid}, \text{out})$  to  $P_i$ .
- **Handling  $\mathcal{G}_{\text{RO}}$  queries**
  1. Initialize empty set  $\mathcal{Q}_{\text{ro}}$ .
  2. Upon receiving input  $(\text{QUERY}, x)$  from a party  $P_i$ , forward it to the  $\mathcal{G}_{\text{RO}}$  and forward the response  $v$  back to  $P_i$ .
  3. Record  $x$  in  $\mathcal{Q}_{\text{ro}}$ .
- **Simulation of uncorrupted prover** Upon receiving input  $(\text{PROVE}, \text{sid}, C)$  from  $\mathcal{F}_{\text{NIZK}}$ :
  1. Derive the witness size  $n$  from the description of  $C$ . Compute  $\ell$  and  $d$  as **Proof** of  $\Pi_{\text{UC-R}}$  would and if  $d > D$  abort by outputting  $(\text{PROOF}, \perp)$ .
  2. Generate a polynomial encoding of dummy witness:  $f \leftarrow \text{Enc}(0^n, n, \ell; \rho_w)$ , where  $\rho_w \leftarrow \mathbb{F}^\ell$ .
  3. Generate a commitment to the polynomial encoding as **Proof** of  $\Pi_{\text{UC-R}}$  would:  $c \leftarrow \text{Com}(\text{ck}, f, d; \rho_c)$ .
  4. Define the circuit  $C'$  as **Proof** of  $\Pi_{\text{UC-R}}$  would.
  5. Run  $\Pi_{\mathcal{R}.S}(2, \text{st}, C')$  to obtain a proof-state pair  $(\pi', \text{st})$ .
  6. Create  $\mathbf{z}, \mathbf{y}$ , and  $\pi_{\text{PCS}}$  as **Proof** of  $\Pi_{\text{UC-R}}$  would.
  7. Send  $(\text{PROOF}, \varpi)$  to  $\mathcal{F}_{\text{NIZK}}$ , where  $\varpi := (\pi', c, \mathbf{z}, \mathbf{y}, \pi_{\text{PCS}})$
- **Simulation of uncorrupted verifier** Upon receiving input  $(\text{VERIFY}, \text{sid}, C, \varpi)$  from  $\mathcal{F}_{\text{NIZK}}$ :
  1. Perform verification checks similar to **Verification** of  $\Pi_{\text{UC-R}}$ , but use  $\text{pp}$  and  $\text{ck}$  generated during the simulation of  $\mathcal{F}_{\text{Setup}}$ . Calls to  $\mathcal{O}_{\text{Setup}}$  made by  $\Pi_{\mathcal{R}.S}$  are handled by running  $(\text{out}, \text{st}) \leftarrow \Pi_{\mathcal{R}.S}(1, \text{st}, \text{in})$  and forwarding  $\text{out}$  to  $\mathcal{V}$ . If invalid, send  $(\text{WITNESS}, \perp_1)$  to  $\mathcal{F}_{\text{NIZK}}$ . This will eventually cause  $\mathcal{F}_{\text{NIZK}}$  to output  $(\text{VERIFICATION}, \text{sid}, \text{ssid}, 0)$  to a dummy party  $\tilde{P}_i$ .
  2. Parse proof  $\varpi$  as  $(\pi', c, \mathbf{z}, \mathbf{y}, \pi_{\text{PCS}})$ .
  3. Query  $\mathcal{G}_{\text{RO}}$  on  $(\text{OBSERVE}, \text{sid})$  and receive the set of illegitimate queries  $\mathcal{Q}_{|\text{sid}}$ .
  4. Update  $\mathcal{Q}_{\text{ro}} = \mathcal{Q}_{\text{ro}} \cup \mathcal{Q}_{|\text{sid}}$ .
  5. Define circuit  $C'$  as **Verification** of  $\Pi_{\text{UC-R}}$  would.
  6. Define  $\mathcal{Q}_c$  as the set of queries in  $\mathcal{Q}_{\text{ro}}$  of the form  $(\text{QUERY}, (\text{sid}, (C', c, \cdot, \cdot, \cdot)))$  such that evaluation proof is valid. If there are more than one queries with the same evaluation point  $z$  then, irrespective of the iteration  $i$ , include only the very first such query in  $\mathcal{Q}_c$ .
  7. In the set  $\mathcal{Q}_c$ , if for the same  $(c, z)$ , there exists  $(y, \pi)$  and  $(y', \pi')$  such that  $y \neq y'$  or  $\pi \neq \pi'$ , then set  $w := \perp_2$  and go to 12.
  8. If  $(C', \pi')$  was previously generated by  $\Pi_{\mathcal{R}.S}$  then set  $w := \perp_3$  and go to 12.
  9. If  $|\mathcal{Q}_c| < d$  then set  $w := \perp_4$  and go to 12.
  10. Otherwise, parse  $\mathcal{Q}_c$  as tuples  $\{(C', c, z_j, y_j, \pi_j, i_j)\}$ , where each  $z_j$  is distinct. Collect polynomial evaluations  $(z_j, y_j)$  and interpolate the polynomial  $f$  of degree  $d - 1$  such that for  $j \in [d]$ ,  $y_j = f(z_j)$ .
  11. If  $(C, \text{Dec}(f)) \notin \mathcal{R}$  set  $w := \perp_5$ ; Else, set  $w := \text{Dec}(f)$ .
  12. Send  $(\text{WITNESS}, w)$  to  $\mathcal{F}_{\text{NIZK}}$ .

**Fig. 5.** Simulator for  $\Pi_{\text{UC-R}}$ .



**Definition 4.1. (SDH Assumption).** The strong Diffie-Hellman assumption (SDH) holds with respect to a bilinear group generator  $\text{BGen}$  if for all PPT adversaries  $\mathcal{A}$  and degree bound  $D > 0$ ,

$$\Pr \left[ t = g^{\frac{1}{\alpha+c}} : \mathcal{G} \leftarrow \text{BGen}(1^\lambda); \alpha \leftarrow_{\$} \mathbb{F}; \sigma := (\{g^{\alpha^i}\}_{i=0}^{D-1}, h^\alpha); (t, c) \leftarrow \mathcal{A}(\mathcal{G}, \sigma) \right] \leq \text{negl}(\lambda)$$

**Lemma 4.2.**  $\text{PCS}_{\text{KZG}}$  is perfectly unique (Definition 2.7), computationally evaluation binding under the SDH assumption, perfectly  $\phi$ -evaluation hiding (Definition 2.9), and computationally  $\phi$ -non-extrapolation (Definition 2.10) with respect to any polynomial encoding scheme PES with bounded independence (Definition 2.8), where  $\phi(r) := r + 1$ .

*Proof. Unique Proof.* We prove there exists unique  $\boldsymbol{\pi} = (\pi, \hat{\pi}, \hat{y})$  for a fixed  $\mathbf{c} = (c, \hat{c}), d, z$ , and  $y$ . Due to the pairing equation, a valid  $\pi$  is uniquely determined by  $(c/g^y)^{\frac{1}{\alpha-z}}$ . The same holds for  $\hat{\pi}$ . Finally, a valid  $\hat{y}$  is uniquely determined by  $z^{D-d}y$ .

**Evaluation Binding.** Suppose the adversary outputs  $\mathbf{c} = (c, \hat{c}), d, z, y, y' \neq y, \boldsymbol{\pi} = (\pi, \hat{\pi}, \hat{y}), \boldsymbol{\pi}' = (\pi', \hat{\pi}', \hat{y}')$  such that both proofs verify. If  $g^z = g^\alpha$ , then SDH is broken with solution  $(g^{1/z}, 0)$ . Otherwise, we have  $(\pi/\pi')^{\frac{1}{y'-y}} = g^{\frac{1}{\alpha-z}}$  thanks to the pairing equation and thus SDH is broken with solution  $((\pi/\pi')^{\frac{1}{y'-y}}, -z)$ .<sup>8</sup>

**Evaluation Hiding.** Let  $r = |\mathbf{z}|$  be the number of evaluations requested by the adversary. Due to the bounded independence of PES, any set of  $\phi(r) = r + 1$  evaluations of encoded polynomial  $f$  in  $\mathcal{D}_{\text{Enc}}$  are independently and uniformly distributed in  $\mathbb{F}$ . The commitment  $\mathbf{c} = (c, \hat{c})$  leaks at most a single evaluation  $f(\alpha)$ . For  $i \in [r]$ , each proof  $(\pi_i, \hat{\pi}_i, \hat{y}_i)$  leaks at most  $f(\alpha)$  and  $f(z_i)$ . Overall, the adversary observes at most  $r + 1$  evaluations of  $f$ , whose distribution is independent and uniform in  $\mathbb{F}$ .

**Non-Extrapolation.** For KZG polynomial commitment scheme used with PES,  $\phi(r) := r + 1$ . We show the following hybrids to prove non-extrapolation.

1. **Hyb<sub>0</sub>**: The same as the game defined in Definition 2.10, i.e., an all-zero vector of length  $n$  is encoded as a polynomial and the adversary  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  is provided with up to  $r$  distinct evaluations plus proofs.
2. **Hyb<sub>1</sub>**: The challenger’s code is changed as: Instead of encoding an all-0 vector, sample  $d$  random evaluations  $y_i \leftarrow_{\$} \mathbb{F}$ . Recall, degree of the encoded polynomial is denoted by  $d - 1$ . Let  $|\mathbf{z}|_{\text{u}}$  denote the number of distinct values in  $\mathbf{z}$ . Let  $r' := |\mathbf{z}|_{\text{u}}$  and  $n' := d - r'$ . Note that, when there are no repeat elements in  $\mathbf{z}$ ,  $r = r'$ . Sample  $n'$  evaluation points from  $\mathcal{D}_{\text{Enc}}^{n'}$  and interpolate the polynomial  $f$  defined by  $d$  points  $(z_i, y_i)$ , where the first  $r'$   $z_i$ ’s are from  $\mathcal{A}_1$ , and the rest are sampled by the challenger. Computing commitments and evaluation proofs is same as before.

<sup>8</sup> Since the reduction only relies on the first component of the proof the scheme even satisfies a slightly stronger variant of evaluation binding where the adversary gets to choose distinct degree bounds for different evaluation proofs.

This hybrid remains indistinguishable from the previous one because of  $\phi$ -evaluation hiding of the PCS scheme. In particular, up to  $r + 1$  distinct evaluations and proofs do not reveal anything about the underlying committed polynomial. For  $i \in [r]$ , each proof leaks at most one evaluation  $f(z_i)$  and  $f(\alpha)$ . Thus, overall the adversary learns at most  $r + 1$  distinct evaluations only.

Now, after the execution of  $\text{Hyb}_1$ ,  $\mathcal{A}_2$  outputs a valid evaluation proof for a new point  $(z^*, y^*, \pi^*)$ . Let  $\tilde{y} = f(z^*)$ . Since, the committed polynomial  $f$  is random and has degree  $d - 1 = n + r$ , and  $\mathcal{A}$  learns at most  $r + 1$ , there is at least one degree of freedom corresponding to which the evaluation of  $f$  is uniformly distributed in  $\mathbb{F}$ . This implies that the probability of  $y^* = \tilde{y}$  is  $1/|\mathbb{F}|$ , which is negligible. In case,  $y^* \neq \tilde{y}$ , the challenger obtains two different evaluations and valid proof for the same point which contradicts evaluation binding for the PCS scheme. Thus,  $\mathcal{A}$  wins only with negligible probability.

## 4.2 Candidate NIZK Schemes

Our compiler lifts any simulation extractable SNARK (SE-SNARK) to a UC NIZK. Plugging in any SE-SNARK therefore yields a UC NIZK under the same assumptions. However, the security analyses of many SNARKs in the literature are in idealized models like the Generic Group Model (GGM) or Algebraic Group Model (AGM) [40], and such analyses do not provide any guarantees outside of those models. As we wish to prove composition with respect to any environment (not just algebraic ones, for instance), the most meaningful candidates to plug into our compiler are those that provide guarantees about *any* adversary, even by making use of (non-black-box) knowledge assumptions.

**Immediately Compatible SE-SNARKs.** Groth and Maller [53] construct an SE-SNARK from a knowledge assumption that they formulate, called the eXtended Power Knowledge of Exponent (XPKE) assumption. Lipmaa [64] presents SE-SNARKs under ‘hash-algebraic’ knowledge assumptions. Abdolmaleki et al. [2] show how to lift any zk-SNARK to an SE-SNARK (with non-black-box extraction), and present a concrete instantiation based on the zk-SNARK of Groth et al. [52], which in turn relies on knowledge assumptions that they introduce. One could of course apply Abdolmaleki et al.’s approach to any  $O_\lambda(1)$ -sized zk-SNARK to obtain a  $O_\lambda(1)$ -sized SE-SNARK under the same knowledge assumptions. All of these SE-SNARKs are  $O_\lambda(1)$ -sized and can be plugged into our compiler to obtain  $O_\lambda(1)$ -sized UC NIZKs with provably secure composition with respect to any environment, under the same knowledge assumptions.

**Future Work: Alternative Instantiations.** While we have been focused on obtaining  $O_\lambda(1)$ -sized UC NIZKs in this paper, our compiler can be more widely applicable. In general, given a NIZK that produces proofs of size  $\mathcal{O}_\lambda(f(|C| + |w|))$  and a polynomial commitment scheme that produces evaluation proofs of size  $\mathcal{O}_\lambda(g(|w|))$  for some functions  $f, g$ , our compiler produces a UC NIZK (in the ROM) where the proofs are of size  $O_\lambda(f(|C| + |w|) + g(|w|))$ , under the same

setup and knowledge assumptions as the NIZK and polynomial commitment. With the right input SNARKs, we can obtain witness-succinct UC NIZKs that have benefits orthogonal to  $O_\lambda(1)$ -sized proofs. Consider the following:

- A ‘transparent’ input SNARK—one that does not require a structured common reference string—would result in a transparent UC NIZK with the same succinctness upon applying our compiler. For instance, the recent work of Arun et al. [4] gives such a constant sized transparent SNARK using class groups, however their analysis is in the generic group model, and simulation extractability of their construction has yet to be analyzed.
- If one were to plug in a SNARK that does not require non-black-box knowledge assumptions, we would obtain a UC NIZK that does not either. For instance, plugging in Bulletproofs [22] into our compiler with a transparent polynomial commitment scheme (in the ROM) would result in a  $O_\lambda(\log(|C| + |w|))$  sized transparent UC NIZK in the ROM alone, that does not rely on any knowledge assumptions, and only assumes the hardness of computing discrete logarithms. One hurdle to overcome for such an instantiation is that while Bulletproofs are known to be simulation extractable in the AGM [44], there is at present no such analysis in the random oracle model alone (to our knowledge).

The scope of this paper is limited to the design and analysis of our general compiler, and so we leave such custom instantiations to future work.

## References

1. Abdalla, M., Barbosa, M., Katz, J., Loss, J., Xu, J.: Algebraic adversaries in the universal composability framework. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 311–341. Springer, Heidelberg, December 2021. [https://doi.org/10.1007/978-3-030-92078-4\\_11](https://doi.org/10.1007/978-3-030-92078-4_11)
2. Abdolmaleki, B., Ramacher, S., Slamanig, D.: Lift-and-shift: obtaining simulation extractable subversion and updatable SNARKs generically. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020, pp. 1987–2005. ACM Press, November 2020. <https://doi.org/10.1145/3372297.3417228>
3. Ames, S., Hazay, C., Ishai, Y., Venkatasubramanian, M.: Ligerio: lightweight sub-linear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 2087–2104. ACM Press, Oct/Nov 2017. <https://doi.org/10.1145/3133956.3134104>
4. Arun, A., Ganesh, C., Lokam, S., Mopuri, T., Sridhar, S.: Dew: transparent constant-sized zkSNARKs. Cryptology ePrint Archive, Report 2022/419 (2022). <https://eprint.iacr.org/2022/419>
5. Atapoor, S., Bagheri, K.: Simulation extractability in groth’s zk-SNARK. Cryptology ePrint Archive, Report 2019/641 (2019). <https://eprint.iacr.org/2019/641>
6. 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](https://doi.org/10.1007/978-3-030-35199-1_3)
7. Bagheri, K., Kohlweiss, M., Siim, J., Volkhov, M.: Another look at extraction and randomization of Groth’s zk-SNARK. In: Borisov, N., Diaz, C. (eds.) FC 2021.

- LNCS, vol. 12674, pp. 457–475. Springer, Heidelberg (2021). [https://doi.org/10.1007/978-3-662-64322-8\\_22](https://doi.org/10.1007/978-3-662-64322-8_22)
8. Baghery, K., Pindado, Z., Ràfols, C.: Simulation extractable versions of groth’s zk-SNARK revisited. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 2020. LNCS, vol. 12579, pp. 453–461. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-65411-5\\_22](https://doi.org/10.1007/978-3-030-65411-5_22)
  9. Baghery, K., Sedaghat, M.: TIRAMISU: black-box simulation extractable NIZKS in the updatable CRS model. In: Conti, M., Stevens, M., Krenn, S. (eds.) CANS 2021. LNCS, vol. 13099, pp. 531–551. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-92548-2\\_28](https://doi.org/10.1007/978-3-030-92548-2_28)
  10. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93, pp. 62–73. ACM Press, November 1993. <https://doi.org/10.1145/168588.168596>
  11. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046 (2018). <https://eprint.iacr.org/2018/046>
  12. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018). <https://eprint.iacr.org/2018/046>
  13. Ben-Sasson, E., Chiesa, A., Gabizon, A., Virza, M.: Quasi-linear size zero knowledge from linear-algebraic PCPs. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016, Part II. LNCS, vol. 9563, pp. 33–64. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49099-0\\_2](https://doi.org/10.1007/978-3-662-49099-0_2)
  14. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent Succinct Arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17653-2\\_4](https://doi.org/10.1007/978-3-030-17653-2_4)
  15. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016, Part II. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53644-5\\_2](https://doi.org/10.1007/978-3-662-53644-5_2)
  16. Blum, M., Feldman, P., Micali, S.: Proving security against chosen ciphertext attacks. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 256–268. Springer, New York (1990). [https://doi.org/10.1007/0-387-34799-2\\_20](https://doi.org/10.1007/0-387-34799-2_20)
  17. Blum, M., Santis, A.D., Micali, S., Persiano, G.: Noninteractive zero-knowledge. *SIAM J. Comput.* **20**(6), 1084–1118 (1991) <https://doi.org/10.1137/0220068>
  18. Boneh, D., Boyen, X.: Efficient selective-id secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-24676-3\\_14](https://doi.org/10.1007/978-3-540-24676-3_14)
  19. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49896-5\\_12](https://doi.org/10.1007/978-3-662-49896-5_12)
  20. Bowe, S., Gabizon, A.: Making groth’s zk-snark simulation extractable in the random oracle model. Cryptology ePrint Archive, Paper 2018/187 (2018). <https://eprint.iacr.org/2018/187>
  21. Bowe, S., Gabizon, A., Green, M.D.: A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. In: Zohar, A., Eyal, I., Teague, V., Clark, J., Bracciali, A., Pintore, F., Sala, M. (eds.) FC 2018. LNCS, vol. 10958,

- pp. 64–77. Springer, Heidelberg (2019). [https://doi.org/10.1007/978-3-662-58820-8\\_5](https://doi.org/10.1007/978-3-662-58820-8_5)
22. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press, May 2018. <https://doi.org/10.1109/SP.2018.00020>
  23. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 1–18. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-64378-2\\_1](https://doi.org/10.1007/978-3-030-64378-2_1)
  24. Camenisch, J., Damgård, I.: Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 331–345. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-44448-3\\_25](https://doi.org/10.1007/3-540-44448-3_25)
  25. Camenisch, J., Drijvers, M., Gagliardoni, T., Lehmann, A., Neven, G.: The wonderful world of global random oracles. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 280–312. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-78381-9\\_11](https://doi.org/10.1007/978-3-319-78381-9_11)
  26. Campanelli, M., Faonio, A., Fiore, D., Querol, A., Rodríguez, H.: Lunar: a toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 3–33. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-92078-4\\_1](https://doi.org/10.1007/978-3-030-92078-4_1)
  27. Campanelli, M., Ganesh, C., Khoshakhlagh, H., Siim, J.: Impossibilities in succinct arguments: Black-box extraction and more. Cryptology ePrint Archive, Report 2022/638 (2022). <https://eprint.iacr.org/2022/638>
  28. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001. <https://doi.org/10.1109/SFCS.2001.959888>
  29. 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](https://doi.org/10.1007/978-3-540-70936-7_4)
  30. 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. <https://doi.org/10.1145/2660267.2660374>
  31. 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. <https://doi.org/10.1145/509907.509980>
  32. Canetti, R., Sarkar, P., Wang, X.: Triply adaptive UC NIZK. Cryptology ePrint Archive, Report 2020/1212 (2020). <https://eprint.iacr.org/2020/1212>
  33. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 738–768. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45721-1\\_26](https://doi.org/10.1007/978-3-030-45721-1_26)
  34. Chiesa, A., Ojha, D., Spooner, N.: FRACTAL: post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 769–793. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45721-1\\_27](https://doi.org/10.1007/978-3-030-45721-1_27)
  35. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-44647-8\\_33](https://doi.org/10.1007/3-540-44647-8_33)

36. Dodis, Y., Shoup, V., Walfish, S.: Efficient constructions of composable commitments and zero-knowledge proofs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 515–535. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-85174-5\\_29](https://doi.org/10.1007/978-3-540-85174-5_29)
37. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography (extended abstract). In: 23rd ACM STOC, pp. 542–552. ACM Press, May 1991. <https://doi.org/10.1145/103418.103474>
38. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the fiat-shamir transform. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-34931-7\\_5](https://doi.org/10.1007/978-3-642-34931-7_5)
39. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 152–168. Springer, Heidelberg (2005). [https://doi.org/10.1007/11535218\\_10](https://doi.org/10.1007/11535218_10)
40. Fuchsbauer, G., Kiltz, E., Loss, J.: The Algebraic Group Model and its Applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 33–62. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-96881-0\\_2](https://doi.org/10.1007/978-3-319-96881-0_2)
41. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019). <https://eprint.iacr.org/2019/953>
42. Ganesh, C., Khoshakhlagh, H., Kohlweiss, M., Nitulescu, A., Zajac, M.: What makes fiat-shamir zkSNARKs (updateable SRS) simulation extractable? In: Galdi, C., Jarecki, S. (eds.) SCN 2022. Lecture Notes in Computer Science, vol. 13409, pp. 735–760. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-14791-3\\_32](https://doi.org/10.1007/978-3-031-14791-3_32)
43. Ganesh, C., Kondi, Y., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Witness-succinct universally-composable snarks. Cryptology ePrint Archive, Paper 2022/1618 (2022). <https://eprint.iacr.org/2022/1618>
44. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 397–426. Springer, Heidelberg, May/June 2022. [https://doi.org/10.1007/978-3-031-07085-3\\_14](https://doi.org/10.1007/978-3-031-07085-3_14)
45. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-shamir bulletproofs are non-malleable (in the random oracle model). Cryptology ePrint Archive, Paper 2023/147 (2023). <https://eprint.iacr.org/2023/147>
46. Garay, J.A., MacKenzie, P., Yang, K.: Strengthening Zero-Knowledge Protocols Using Signatures. *J. Cryptology* **19**(2), 169–209 (2005). <https://doi.org/10.1007/s00145-005-0307-3>
47. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-38348-9\\_37](https://doi.org/10.1007/978-3-642-38348-9_37)
48. Goldreich, O., Håstad, J.: On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.* **67**(4), 205–214 (1998)
49. Goldreich, O., Vadhan, S., Wigderson, A.: On interactive proofs with a laconic prover. *Comput. Complexity* **11**(1), 1–53 (2002)
50. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 444–459. Springer, Heidelberg (2006). [https://doi.org/10.1007/11935230\\_29](https://doi.org/10.1007/11935230_29)

51. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49896-5\\_11](https://doi.org/10.1007/978-3-662-49896-5_11)
52. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 698–728. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-96878-0\\_24](https://doi.org/10.1007/978-3-319-96878-0_24)
53. Groth, J., Maller, M.: Snarky signatures: minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 581–612. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63715-0\\_20](https://doi.org/10.1007/978-3-319-63715-0_20)
54. 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](https://doi.org/10.1007/11761679_21)
55. Groth, J., Ostrovsky, R., Sahai, A.: New techniques for noninteractive zero-knowledge. *J. ACM* **59**(3), 11:1–11:35 (2012). <https://doi.org/10.1145/2220357.2220358>
56. Jain, A., Pandey, O.: Non-malleable zero knowledge: black-box constructions and definitional relationships. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 435–454. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-10879-7\\_25](https://doi.org/10.1007/978-3-319-10879-7_25)
57. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-17373-8\\_11](https://doi.org/10.1007/978-3-642-17373-8_11)
58. Katsumata, S.: A new simple technique to bootstrap various lattice zero-knowledge proofs to QROM secure NIZKs. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 580–610. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84245-1\\_20](https://doi.org/10.1007/978-3-030-84245-1_20)
59. Kattis, A., Panarin, K., Vlasov, A.: RedShift: transparent SNARKs from list polynomial commitment IOPs. Cryptology ePrint Archive, Report 2019/1400 (2019). <https://eprint.iacr.org/2019/1400>
60. Kerber, T., Kiayias, A., Kohlweiss, M.: Composition with knowledge assumptions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 364–393. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84259-8\\_13](https://doi.org/10.1007/978-3-030-84259-8_13)
61. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC, pp. 723–732. ACM Press, May 1992. <https://doi.org/10.1145/129712.129782>
62. Kondi, Y., Shelat, A.: Improved straight-line extraction in the random oracle model with applications to signature aggregation. Cryptology ePrint Archive, Report 2022/393 (2022). <https://eprint.iacr.org/2022/393>
63. Kosba, A., et al.:  $C\theta c\theta$ : a framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093 (2015), <https://eprint.iacr.org/2015/1093>
64. Lipmaa, H.: Simulation-extractable SNARKs revisited. Cryptology ePrint Archive, Report 2019/612 (2019). <https://eprint.iacr.org/2019/612>
65. Lysyanskaya, A., Rosenbloom, L.N.: Efficient and universally composable non-interactive zero-knowledge proofs of knowledge with security against adaptive corruptions. Cryptology ePrint Archive, Paper 2022/1484 (2022). <https://eprint.iacr.org/2022/1484>

66. Lysyanskaya, A., Rosenbloom, L.N.: Universally composable sigma-protocols in the global random-oracle model. Cryptology ePrint Archive, Report 2022/290 (2022). <https://eprint.iacr.org/2022/290>
67. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2111–2128. ACM Press, November 2019. <https://doi.org/10.1145/3319535.3339817>
68. Maurer, U.: Constructive cryptography – a new paradigm for security definitions and proofs. In: Mödersheim, S., Palamidessi, C. (eds.) TOSCA 2011. LNCS, vol. 6993, pp. 33–56. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-27375-9\\_3](https://doi.org/10.1007/978-3-642-27375-9_3)
69. Micali, S.: Computationally sound proofs. SIAM J. Comput. **30**(4), 1253–1298 (2000) <https://doi.org/10.1137/S0097539795284959>
70. 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. <https://doi.org/10.1109/SP.2013.47>
71. Pass, R.: On deniability in the common reference string and random oracle model. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 316–337. Springer, Heidelberg (2003). [https://doi.org/10.1007/978-3-540-45146-4\\_19](https://doi.org/10.1007/978-3-540-45146-4_19)
72. Pass, R., Rosen, A.: New and improved constructions of non-malleable cryptographic protocols. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC, pp. 533–542. ACM Press, May 2005. <https://doi.org/10.1145/1060590.1060670>
73. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th FOCS, pp. 543–553. IEEE Computer Society Press, October 1999. <https://doi.org/10.1109/SFFCS.1999.814628>
74. Unruh, D.: Non-interactive zero-knowledge proofs in the quantum random oracle model. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 755–784. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46803-6\\_25](https://doi.org/10.1007/978-3-662-46803-6_25)





# Speed-Stacking: Fast Sublinear Zero-Knowledge Proofs for Disjunctions

Aarushi Goel<sup>1</sup>, Mathias Hall-Andersen<sup>2(✉)</sup>, Gabriel Kaptchuk<sup>3</sup>,  
and Nicholas Spooner<sup>4</sup>

<sup>1</sup> NTT Research, Tokyo, Japan

`aarushi.goel@ntt-research.com`

<sup>2</sup> Aarhus University, Aarhus, Denmark

`ma@cs.au.dk`

<sup>3</sup> Boston University, Boston, USA

`kaptchuk@bu.edu`

<sup>4</sup> University of Warwick, Coventry, UK

`Nicholas.Spooner@warwick.ac.uk`

**Abstract.** Building on recent *compilers* for efficient disjunctive composition (*e.g.* an OR of multiple clauses) of zero-knowledge proofs (*e.g.* Goel *et al.* [EUROCRYPT’22]) we propose a new compiler that, when applied to sublinear-sized proofs, can result in sublinear-size disjunctive zero-knowledge with *sublinear* proving times (without meaningfully increasing proof sizes). Our key observation is that simulation in sublinear-size *zero-knowledge* proof systems can be much faster (both concretely and asymptotically) than the honest prover. We study applying our compiler to two classes of  $O(\log n)$ -round protocols: interactive oracle proofs, specifically Aurora [EUROCRYPT’19] and Fractal [EUROCRYPT’20], and folding arguments, specifically Compressed  $\Sigma$ -protocols [CR-YPTO’20, CRYPTO’21] and Bulletproofs [S&P’18]. This study validates that the compiler can lead to significant savings. For example, applying our compiler to Fractal enables us to prove a disjunction of  $\ell$  clauses, each of size  $N$ , with only  $O((N + \ell) \cdot \text{polylog}(N))$  computation, versus  $O(\ell N \cdot \text{polylog}(N))$  when proving the disjunction directly. We also find that our compiler offers a new lens through which to understand zero-knowledge proofs, evidenced by multiple examples of protocols with the same “standalone” complexity that each behave very differently when stacked.

## 1 Introduction

Zero-knowledge proofs and arguments [30] allow a prover to convince the verifier of the validity of an NP statement without revealing anything beyond the validity itself. Early results established that such protocols exist for all NP languages [29], and recent work has proposed zero-knowledge proofs that are more practically efficient [10, 12, 19, 31, 34, 36, 37]. Many of these efficient zero-knowledge proofs are now being used in practice [9, 23, 48], and zero-knowledge proofs have become a critical component of constructing larger cryptographic systems.

**Disjunctive Zero-Knowledge.** A *disjunctive statement* is an NP statement consisting of a logical OR of a set of clauses. We refer to zero-knowledge proofs

optimized for disjunctive statements as “disjunctive zero-knowledge”. Disjunctive zero-knowledge is central to privacy-preserving systems where revealing *which* clause a prover has a witness for might reveal their identity. Disjunctive zero-knowledge has received a great deal of attention [1, 22, 24] and recently there has been renewed interest in optimizing cryptographic protocols for disjunctions, both in the context of zero-knowledge [4, 7, 21, 26, 32, 34, 41] and secure multiparty computation [27, 33, 35].

The simplest approach to disjunctive zero-knowledge is to appeal to NP-completeness: a disjunction of NP statements is itself an NP statement which can be proved using a proof system for NP. In practice, however, this has two key drawbacks: first, the individual clauses may be of a special form that admits efficient zero-knowledge proofs (e.g. a discrete-log relation) but that structure can not be preserved under disjunction. Second, even if the clauses are general circuits, if the clauses are distinct then the resulting circuit is as large as the sum of the size of individual clauses. As a result, the complexity of the proof system grows at least linearly in the number of clauses.

In light of this, one alternative approach that has been explored in the literature is to manually modify *specific* zero-knowledge protocols directly [4, 32, 34] such that they naturally support disjunctive statements. Excitingly, recent work has shown that manual modification can result in protocols with communication sub-linear in the number of clauses [4, 34]. However, such approaches rely strongly on the structure of individual protocols and do not necessarily generalize.

A more robust approach is to build *disjunctive compilers* [1, 7, 22, 26], generic approaches that automatically transform large classes of zero-knowledge protocols into disjunctive zero-knowledge protocols. The seminal work in this area is [22], which proposed an approach that compiled  $\Sigma$ -protocols for disjunctions by having the prover *simulate* the clauses for which it did not have a witness. More recently, Baum et al. [7] and Goel et al. [26] built upon this idea to compile large classes of zero-knowledge protocols into disjunctive zero-knowledge protocols with communication complexity sub-linear in the number of clauses.

**Succinct Proofs.** A proof system is succinct if its communication cost is polylogarithmic in the size of the computation being proven. Succinct zero-knowledge proofs are the subject of a long and active line of research ([11, 12, 17, 25, 31, 39] and many others) and in recent years have become efficient enough to use in practice. Many such proof systems support some expressive NP-complete problem, e.g. arithmetic circuit satisfiability. This raises a natural suggestion: to prove a disjunctive statement, one could simply construct a circuit for the disjunction and employ a succinct proof system. The size of the resulting proof would be only slightly larger than a proof for a single clause.

The main caveat is that, while the proof size is essentially unaffected, the time and space complexities of the prover increase by at least a *multiplicative* factor of the number of distinct clauses, compared to the cost of proving a single clause. Since succinct proof systems typically have quite high prover complexity, avoiding this increase would result in significant savings.

**Stacking Succinct Proofs.** In our work, we explore how we can apply the frameworks developed in recent research on minimizing the communication complexity

of disjunctive zero-knowledge (specifically [26]) to achieve succinct proofs for disjunctions which avoid this multiplicative blowup in the prover computation time.

At the heart of our approach is the observation that succinct proof systems often have faster simulators than provers. Intuitively, this is because the cost of “cheating” the verifier in a zero-knowledge protocol generally scales with the verifier’s running time, rather than the prover’s. Thus, following the approach of Cramer, Damgård and Schoenmakers, [22], the prover in a succinct proof system can run the (more efficient) simulator for the inactive clauses instead of the (less efficient) prover algorithm.

Taken together, we obtain succinct proof systems that can prove disjunctions without incurring a multiplicative increase in prover complexity in the number of clauses. We also show that in some cases, we can also avoid a similar increase in the *verifier’s* complexity using batching techniques.

**Set Membership vs. True Disjunctions.** There is an important case in which appealing to NP completeness *is* concretely efficient: specifically, if (1) the zero-knowledge protocol supports an expressive NP-complete language, and (2) there is a high degree of homogeneity between the clauses. If a prover wants to prove e.g.  $\mathbf{x}_1 \in \mathcal{L} \vee \dots \vee \mathbf{x}_\ell \in \mathcal{L}$ , it can do so efficiently by proving the statement “ $\exists(i, \mathbf{x}), \text{st. } \mathbf{x} \in \mathcal{L} \wedge \mathbf{x} = \mathbf{x}_i$  (so the choice of branch is part of the NP witness). The size of this circuit is only slightly larger than the circuit for  $\mathcal{L}$  itself. We refer to such statements as *set membership statements*.

Our results are most significant in the case of what we call *true disjunctions*, i.e., where the prover wants to prove e.g.  $\mathbf{x}_1 \in \mathcal{L}_1 \vee \dots \vee \mathbf{x}_\ell \in \mathcal{L}_\ell$  making the above transformation more expensive. In addition to being a more technically challenging statement structure, true disjunctions are also important for many applications. For example, Heath and Kolesnikov studied showing the existence of a bug in a code base [34] in zero-knowledge, which implicitly embeds a true disjunction. It is easy to imagine many other such applications: a prover could want to demonstrate that a image is the product of applying one of a number of sanctioned image modification algorithms (eg. blur, red eye, etc. . . ) to some committed photograph, or a financial institution might want to demonstrate that a transaction satisfies one of a number of policies that would make it compliant.

## 1.1 Our Contributions

### **Framework for Prover-Efficient Succinct Disjunctive Zero-Knowledge.**

We present a framework, which we refer to as *speed stacking*, for composing succinct proofs for disjunctions that often yield significant improvements in prover time. We do this by extending the notion of a “stackable”  $\Sigma$ -protocol, introduced by Goel et al. [26], to a more general notion of a “stackable” interactive protocol. At a high level, a protocol is stackable if it has a zero-knowledge simulator which can be decomposed into a randomized, statement-independent part  $\mathcal{S}_{\text{RAND}}$ , and a deterministic part  $\mathcal{S}_{\text{DET}}$  that completes the work of  $\mathcal{S}_{\text{RAND}}$  for some specific statement. We then show how to compile a stackable zero-knowledge interactive protocol (ZK-IP) into a disjunctive zero-knowledge interactive protocol. Specifically, we prove the following theorem:

**Theorem 1 (Informal).** *Let  $\Pi$  be a “stackable” zero-knowledge interactive protocol for a NP relation  $\mathcal{R}$  with associated simulator  $\mathcal{S}$ . Then, there exists a zero-knowledge interactive protocol  $\Pi'$  for the NP relation  $\mathcal{R}'((\mathbf{x}_1, \dots, \mathbf{x}_\ell), \mathbf{w}) := \exists i, \mathcal{R}_i(\mathbf{x}_i, \mathbf{w}) = 1$  with communication complexity proportional to  $\mathbb{C}(\Pi) + O(\log(\ell))$  and prover computational complexity  $\text{Time}(\Pi) + (\ell - 1) \cdot \text{Time}(\mathcal{S})$ .*

This theorem covers *true disjunctions* when  $\mathcal{R}$  is sufficiently expressive, e.g.

$$\mathcal{R} = \text{circuit-SAT} : \mathcal{R}'(((C_1, \mathbf{x}_1), \dots, (C_\ell, \mathbf{x}_\ell)), \mathbf{w}) = \exists i, C_i(\mathbf{x}_i, \mathbf{w}) = 1.$$

Note that while the above is a “universal” relation, our approach does not make use of universal circuits. As we discuss in the technical overview, while universal circuits are conceptually elegant (and sometimes achieve good asymptotic efficiency), the associated overhead makes them impractical.

Next, we study the speed-stackability of two protocols from each of two families of sublinear-sized zero-knowledge proof systems: interactive oracle proofs and folding arguments. Interestingly, we find that the concrete savings offered by each of the four protocols we consider differ dramatically, offering anything from significant, asymptotic speed-ups to concrete savings without asymptotic gains to minimal speedups. In addition to the new protocols we design, these results offer a new lens through which to study zero-knowledge proofs.

**Speed Stacking Interactive Oracle Proofs.** We adapt our stackability framework to interactive oracle proofs (IOPs) [11], a generalization of interactive proofs that underlies various efficient succinct argument constructions. We show how to adapt the [11, 20] transformations to convert stackable IOPs (resp. holographic IOPs) into stackable succinct arguments (resp. with preprocessing).

We then consider the stackability of two existing IOP protocols for the *rank-one constraint satisfaction* (R1CS) language. Let  $\ell$  be the number of clauses and  $N$  be the maximum circuit size of a clause.

- **Aurora** [10] can be easily seen to be efficiently stackable by carefully examining the zero-knowledge simulator. By applying our compiler, we obtain a stackable succinct argument where the prover runs in time  $O_{\mathbb{F}}(N(\log N + \ell \log^2 \lambda \log \log \lambda))$ . By comparison, the cost of directly proving a disjunction using Aurora is  $O_{\mathbb{F}}(\ell N \log(\ell N))$ .<sup>1</sup>
- **Fractal** [20] is not itself efficiently stackable: the verifier runs in polylogarithmic time after preprocessing, whereas any simulator for the original Fractal protocol involves a linear-time statement-dependent computation. To address this, we modify Fractal into a protocol we call **Stactal**, a stackable IOP for R1CS with polylogarithmic simulation. By applying our compiler, we obtain a stackable succinct argument where the prover runs in time  $O_{\mathbb{F}}(N \log N + \ell \cdot \text{polylog}(N))$ . In particular, proving a disjunction on  $\ell$  clauses for  $\ell \ll N$  is asymptotically as efficient as proving a single clause.

**Speed Stacking Folding Arguments.** Finally, we show how to apply our framework to “folding arguments” [3–5, 17, 19]. This class of protocols, best represented by Compressed  $\Sigma$ -protocols [4] and Bulletproofs [19], in which the prover

<sup>1</sup>  $O_{\mathbb{F}}$  indicates that time complexity is measured in field operations.

replaces a linear-sized protocol message in a zero-knowledge interactive proof with a multi-round, privacy-free, interactive protocol with logarithmic communication complexity.

- **Compressed  $\Sigma$ -Protocols** [3, 4] is a stackable ZK-IP for openings of linear forms (after very minor modifications).

By applying our compiler, we obtain a ZK-IP for the disjunction of linear form openings in which simulating each additional clause only requires computing one exponentiation and one group multiplication, in addition to a linear number of field operations. We also show that our ideas extend to the circuit-satisfiability variant of compressed  $\Sigma$ -protocols. We note that our results are stronger than the *set membership* version of Compressed  $\Sigma$ -protocols presented by Attema et al. [4] in that our approach supports *true disjunctions* as well.<sup>2</sup>

- **Bulletproofs** [19] We observe that Bulletproofs (both for range proofs and circuit satisfiability) are stackable. However, we note that the runtime of the simulator for bulletproofs is roughly the same as that of the prover. As such, speed-stacking bulletproofs provides only marginal benefits over more direct techniques. The only exception we note is proving *set-membership* range proofs; because the range proof version of bulletproofs is not sufficiently expressive to directly capture set-membership, speed-stacking is preferable to rephrasing the statement to circuit satisfiability. This presents an interesting contrast between Compressed  $\Sigma$ -protocols and Bulletproofs, which otherwise seem to rely on very similar techniques.

## 2 Technical Overview

### 2.1 Disjunctive Templates for Zero-Knowledge

Given a sequence of statements  $(\mathfrak{x}_1, \dots, \mathfrak{x}_\ell)$ , we wish to prove in zero-knowledge that either  $\mathfrak{x}_1 \in \mathcal{L}_1$ ,  $\mathfrak{x}_2 \in \mathcal{L}_2$ ,  $\dots$ , or  $\mathfrak{x}_\ell \in \mathcal{L}_\ell$ . While we might have access to appropriate and efficient zero-knowledge proof systems for each individual language  $\mathcal{L}_1, \dots, \mathcal{L}_\ell$ , it is not clear how to apply these to the disjunction, while ensuring zero-knowledge. Let  $\mathfrak{a}$  denote the clause for which the prover has a witness (the *active* clause). We will refer to the other clauses as *inactive*.

There are two main templates for disjunctive zero-knowledge in the literature:

- (1) *Statement Combination*: Combine the statements to define a new  $\mathcal{L}$  with the relation  $\mathcal{R}((\mathfrak{x}_1, \dots, \mathfrak{x}_\ell), \mathfrak{w}) := \mathcal{R}_1(\mathfrak{x}_1, \mathfrak{w}) \vee \dots \vee \mathcal{R}_\ell(\mathfrak{x}_\ell, \mathfrak{w})$ . and use any existing zero-knowledge proof protocol  $\Pi$  that supports general NP statements.
- (2) *Simulation of Inactive Clauses*: Initially suggested by Cramer, Damgård, and Schoenmakers [22], this approach has been explored primarily in the context of  $\Sigma$ -protocols. In this template, the prover uses the honest prover algorithm for the active clause, and “cheats” by using the *zero-knowledge simulator* for each

<sup>2</sup> We expand on the distinction between set membership and true disjunctions in the next section.

of the inactive clauses. The protocol guarantees that the prover can cheat for *all but one* of the clauses.

The best choice of template depends heavily on the underlying zero-knowledge protocol and the structure of the clauses. If the protocol is not for an NP-complete language (e.g. Schnorr’s protocol [46]), it may be impossible to combine the statements without protocol modifications, making the *simulation* template more attractive. When statement combination is possible, the efficiency of the combination often depends on the homogeneity of the clauses, *i.e.* if it is more like *set membership* or a *true disjunctions*.

Of course, this difference is qualitative, rather than quantitative. Notably, a proof system for *set membership* can be used to construct a *true disjunction* by using universal circuits and a set membership over the programming of the circuit. However, transformations with universal circuits are notoriously expensive: for example, an implementation [42] of Valiant’s UC [47] shows that for a circuit implementing AES in 33,616 gates the universal circuit capable of simulating it has 11,794,323 gates (with 3,135,833 multiplications)—an increase of  $\approx 300\times$ . Although there have been recent improvements on Valiant’s initial constants [43], boolean UCs remain orders of magnitude larger than the circuits they can simulate, and arithmetic UCs would incur even higher constants [42].

**Disjunctive Templates for Succinct Proofs.** We now turn our attention to the disjunctive composition of succinct proofs for NP. We first observe that succinctness by itself implies communication-efficient disjunctive composition via statement combination. Specifically, if the size of the relation circuit is increased by a multiplicative factor of  $\ell$ , a logarithmic-sized proof will only increase in size by an additive factor of  $\log(\ell)$ , resulting in a proof that is only marginally larger.

While communication efficient, this approach, however, increases the running time of the prover by at least a multiplicative factor  $\ell$  (potentially more, depending on the complexity of the proof system). This is of special concern for succinct proof systems where the running time of the prover is often a bottleneck. In addition, many succinct proof systems have *space* complexity which grows linearly in the size of the circuit; in this case, the space requirements also increase by a factor  $\ell$ .<sup>3</sup>

The use of the *simulation* template in the sublinear setting has not yet been explored. We make the following initial observations:

- *Faster Simulators Means Faster Prover Time:* The key feature of the *simulation* template is the use of the simulator for each of the inactive clauses. While the runtime of a simulator is typically proportional to the runtime of the prover in linear-sized zero-knowledge protocols, in sublinear-sized proofs it is common to have simulators that are more efficient—either asymptotically or concretely—than the prover.<sup>4</sup> This observation means that applying

<sup>3</sup> We note that there do exist techniques generic techniques to minimize space complexity of provers, *e.g.* [14, 15].

<sup>4</sup> A similar observation was recently used in a concurrent and independent work of Kim et al. [40] for designing efficient non-malleable zero-knowledge proofs.

the *simulation* template to sublinear-sized zero-knowledge proofs could produce disjunctive composition techniques that do not require the prover to pay—from a computational perspective—for the inactive clauses, resulting in significantly faster (and more space-efficient) provers than those produced by applying the *statement combination* template.

- *Communication Overhead Can Be Avoided*: The seminal construction of [22] yields a protocol whose communication complexity is linear in  $\ell$ . In a recent work, Goel et al. [26] proposed a new instantiation of the *simulation* template for  $\Sigma$ -protocols that can achieve the same results while only introducing an additive term in  $\log(\ell)$  to the proof size. At a high level, they observe that it is possible to simulate the inactive clauses such that they share a third round message with the active clause. When simulation is carried out in this way, there is no need for the prover to send transcripts for each clause, removing the communication overhead of [22].

Taken together, these observations facilitate the “the best of both worlds:” concrete computational savings for the prover without incurring any meaningful communication overhead. However, it is not immediately clear how to mobilize these observations into a concrete protocol proposal. In the paragraphs that follow, we summarize the approach of Goel et al. [26] and then proceed to discuss sublinear-sized proofs.

## 2.2 Stacking Sigmas for Sublinear-Sized Proofs

**The Approach of Stacking Sigmas** [26]. Goel et al. [26] propose a new instantiation of the *simulation* template. Their compiler applies to  $\Sigma$ -protocols (three-round public coin zero-knowledge protocols) that have the following two properties (such  $\Sigma$ -protocols are called *stackable*  $\Sigma$ -protocols in their work):

1. *Recyclable Third Round Messages*: The distribution of the third round message (not conditioned on the first round message) across all instances must be the same. That is, there exists an efficient randomized algorithm that can produce a third round message from the correct distribution. Critically, this algorithm must be independent of the statement.
2. *Deterministic Transcript Completion*: The protocol supports a *deterministic simulator*  $\mathcal{S}_{\text{DET}}$  that can produce an accepting first round message when supplied with a challenge and an arbitrary third round message (from the third round message distribution). Importantly this simulator must be deterministic, as it will be run locally by both the prover and the verifier.

Their compiler is based on a *1-out-of- $\ell$  partially binding commitment scheme*, a vector commitment scheme that is only binding in a single (pre-selected) index. First the prover generates the first round message for the active clause  $a_{\mathbf{a}}$  honestly. Instead of directly sending this message, the prover instead commits to a vector containing  $a_{\mathbf{a}}$  in the  $\mathbf{a}^{\text{th}}$  position and zeros in all other positions such that the binding position is  $\mathbf{a}$ . The verifier then sends a challenge  $c$  to the prover as normal. Next, the prover generates the third round message for the active clause

$z_a$ . Rather than generate a separate third round message for the inactive clauses, the prover instead *reuses*  $z_a$  as the third round message for all clauses. To do this, the prover uses the special deterministic simulator  $\mathcal{S}_{\text{DET}}$  to produce  $a_i$  such that  $a_i, c, z_a$  is an accepting transcript for the statement  $\mathfrak{x}_i$ . The prover’s final message consists of  $z_a$  along with the randomness used to open the 1-out-of- $\ell$  partially binding commitment scheme to the vector  $(a_1, \dots, a_\ell)$ . The verifier is then able to recompute the values  $a_i$  independently, checks that each transcript is accepting, and makes sure that the commitment matches.<sup>5</sup>

**Stackable Zero-Knowledge Interactive Protocols.** In order to apply the *simulation* template to multi-round protocols, we must first extend Goel et al.’s notion of stackability to the multi-round setting (i.e., more than three-round setting). We extend the notion of recyclable messages so that it naturally applies to multi-round protocols. Goel et al. consider the distribution of third round messages with respect to the statement, we define a more fine-grained notion that considers the *joint distribution* of *parts* of multiple prover messages (i.e., messages sent across different rounds) with respect to the statement. That is, we let a part of each prover message be considered *recyclable*, in that it can be *re-used* across multiple statements. In order to be considered recyclable, it must be possible to design a randomized simulator  $\mathcal{S}_{\text{RAND}}$  that can produce these messages independently of the statement. The deterministic simulator  $\mathcal{S}_{\text{DET}}$  can then “complete the transcript,” by computing the remaining, statement-dependent parts of each message. We note that identifying the recyclable component of each prover message is up to the protocol designer and it may be possible to produce multiple recyclable message sets for any given protocol.

**Stacking Multi-round Protocols.** To stack multi-round zero-knowledge interactive protocols, we begin by partitioning each prover message of the protocol into two parts: a recyclable part  $m_{\text{RAND},i}$  and a deterministic completion  $m_{\text{DET},i}$ . The prover then runs a modified version of the original prover for the active clause. When the prover would send a recyclable part of a message, it simply sends the message directly. When the prover would send a non-recyclable message, it instead uses a 1-out-of- $\ell$  binding commitment scheme to commit to a vector containing the message in the active clause’s index. In the final round of the protocol, the prover uses the deterministic simulator to compute the “missing” non-recyclable messages for the inactive clauses and opens all of the commitments.

### 2.3 Speed-Stacking Interactive Oracle Proofs

A key technique for obtaining sublinear-sized interactive arguments is the cryptographic “compilation” of interactive oracle proofs (IOPs) [11, 38, 44, 45]. An interactive oracle proof is an interactive proof system where the verifier, rather

---

<sup>5</sup> Note that to compile the resulting protocol with Fiat-Shamir, the prover passes the *partially-binding commitment* into the random oracle, as the challenge cannot depend on first-round messages that have not yet been computed.



than reading the messages it receives in their entirety, has oracle access to each message and can query the messages at any index. IOPs can be viewed as a natural multi-round generalization of the notion of *probabilistically checkable proof* (PCP) [6]. All IOPs discussed in this paper will be public-coin. A zero-knowledge IOP additionally has an efficient simulator: given the verifier’s random tape, the simulator computes oracle responses to the verifier’s queries which have the same distribution as in the real interaction. Given a succinct vector commitment scheme (e.g. a Merkle tree), an IOP can be transformed into a succinct interactive argument as follows [11]: in each round, the prover simply computes a commitment to the message and sends the commitment to the verifier; the verifier then responds with the set of query points and the prover provides opening proofs for the responses.

In this section we give an overview of our results on the stackability of IOP-based succinct arguments. We provide a two-part framework: we first define a notion of stackability for IOPs, and then show how a stackable IOP can be “compiled” into a stackable interactive argument — with some minor tweaks the existing compiler outlined above preserves “stackability”. We show that several interactive oracle proofs (IOPs) are stackable, specifically Aurora [10] and a variant of Fractal [20] that we call Stactal. Finally, we outline why it is possible to achieve prover computational savings when compiling these protocols. What follows is an informal description of the definitions and techniques described formally in Sect. 4. The central definition is the notion of a “stackable IOP”:

**Stackable IOPs.** A stackable IOP is a zero-knowledge IOP with a particular simulation strategy: there exists a partition of the  $k$  oracles (rounds) into  $R_{\text{rec}}$  and  $[k] \setminus R_{\text{rec}}$ , such (1) responses to queries for oracles in  $R_{\text{rec}}$  can be sampled *independently* from the relation/statement. (2) while responses to queries for oracles in  $[k] \setminus R_{\text{rec}}$  can be computed *deterministically* from the relation/statement and other query answers.

Intuitively a stackable IOP enables reusing the same oracles in  $R_{\text{rec}}$  to simulate multiple IOPs for distinct relations/statement, while communicating the responses for the remaining (distinct) oracles in  $[k] \setminus R_{\text{rec}}$  requires no additional communication – since the expected responses can be deterministically computed by the verifier (by running the simulator).

**Stackable IOPs to Stackable IPs.** Analogously to the way that IOPs can be compiled into arguments in the plain model, stackable IOPs can be compiled into stackable arguments in the plain model. We show that the existing IOP to IP compiler (outlined above) from vector commitments, can be adapted to preserve the efficient “stackability” of the underlying IOP. In order to preserve efficient simulation for the inactive clauses we need the vector commitment scheme to allow committing to and opening a subvector in time that depends only on the size of the subvector. We show that specific instantiations of Merkle trees satisfy this requirement.

**Efficiency.** One of the advantages of IOPs over other sublinear-sized proofs is that the running time of the IOP verifier can be *polylogarithmic* in the size of

the statement. To maintain this property when applying our stacking compiler, we also require that the (instance-dependent component of the) simulator be similarly efficient. This is typically not a design goal for simulators, since polynomial (rather than polylogarithmic) efficiency suffices for zero-knowledge. As such, the security proofs of many existing protocols construct simulators which are not efficient enough for us. In some cases, all that is required is a more careful simulator construction. In others, to achieve efficient simulation we must substantially modify the protocol itself.

**Showing Stackability.** Many IOP constructions share a similar basic structure, consisting of two main parts: an encoded protocol, where soundness holds assuming that the prover’s messages are close to words in an error-correcting code, and a proximity test, which guarantees that this condition holds. The code of choice for most constructions is the Reed–Solomon code, the code of evaluations of low-degree univariate polynomials over finite field  $\mathbb{F}$  on some domain  $L \subseteq \mathbb{F}$ . Achieving zero-knowledge for protocols constructed in this way typically involves only two techniques:

- (1) **Bounded independence:** when the prover sends an encoding of a secret vector  $v \in \mathbb{F}^k$ , rather than directly encoding  $v$ , it chooses a random vector  $r \in \mathbb{F}^b$  and encodes  $v \parallel r \in \mathbb{F}^{k+b}$ . The properties of the Reed–Solomon code guarantee that, under a mild condition on the evaluation domain  $L$ , the answers to any set of  $b$  queries to a codeword are distributed uniformly at random in  $\mathbb{F}$  (that is, the code is  $b$ -wise independent). To simulate, the simulator simply answers any verifier query uniformly at random.
- (2) **Masking:** often the verifier needs to check some linear property  $P$  with respect to the prover’s messages (a property  $P \subseteq \mathbb{F}^\ell$  is linear if it is an  $\mathbb{F}$ -linear subspace of  $\mathbb{F}^\ell$ ). Examples of such properties include the Reed–Solomon code itself (low-degree testing), or the subcode of the Reed–Solomon code consisting of polynomials whose evaluations over a set  $S \subseteq \mathbb{F}$  sum to zero (univariate sumcheck).

Linear properties allow for zero-knowledge via random self-reduction: to show that  $f \in P$ , the prover sends a uniformly random word  $r \in P$  (the “mask”), the verifier chooses a challenge  $\alpha \in \mathbb{F}$  uniformly at random, and the prover and verifier then engage in a protocol to show that  $\alpha f + r \in P$ . To simulate, the simulator first generates a transcript showing that  $q \in P$  for uniformly random  $q \in P$ ; it then answers queries to  $r$  by “querying”  $q - \alpha \cdot f$ . Note that this simulation strategy requires that the simulator can simulate some number of queries to  $f$ , which is typically achieved through bounded independence as described above.

These two techniques lend themselves to the [26] stacking approach, as follows. Simulation for (1) is trivially instance-independent (recyclable), since the simulator simply answers queries uniformly at random. For (2), observe that provided  $P$  is an instance-independent property, and the process of sampling a protocol transcript showing that  $q \in P$  is also instance-independent. Given  $q$ , queries to the mask  $r$  can then be answered deterministically. Hence for essentially all

zero-knowledge IOP constructions, every message is fully recyclable except for those in which the prover sends a random mask.

To demonstrate the above approach, we consider two key IOP constructions from the literature: Aurora [10] and Fractal [20]. We start with the more complicated case of Fractal:

**Fractal/Stactal.** Fractal is a Holographic IOP, which means it can be compiled to a preprocessing zkSNARK in which the verifier’s running time is polylogarithmic. Unfortunately Fractal is *not* an efficiently stackable IOP. The challenge originates in the Fractal “holographic lincheck” which proves, for encodings of (secret) vectors  $x, y$ , that  $Mx = y$  for a public matrix  $M$  that is “holographically” encoded. The central problem with this lincheck is that it reduces to opening a bivariate polynomial  $u_M(\beta, \alpha)$  at a random  $\beta, \alpha \in \mathbb{F}$ . Since this evaluation depends (deterministically) on every nonzero entry of  $M$ , simulation requires reading all of  $M$  to compute the correct  $u_M(\alpha, \beta)$ . As a result, the stacked verifier becomes inefficient. To alleviate this we introduce “Stactal”, a variant of “Fractal” which *does* admit very efficient stacking. “Stactal” modifies the lincheck protocol to allow the prover to extend the matrix  $M$  to a larger matrix  $M'$  that is “padded” with random values. This introduces sufficient bounded independence that  $u_{M'}(\beta, \alpha)$  is uniformly random (and so independent of  $M$ ).

**Aurora.** Aurora is naturally a stackable IOP: since the verifier in Aurora is quasi-linear, the stacking simulator has enough “computational budget” to read all of  $M$ . Hence, the (simpler, non-holographic) lincheck of Aurora can easily be simulated with the same time complexity as the verifier.

## 2.4 Speed-Stacking Folding Arguments

The next class of sublinear-sized zero-knowledge proofs are ones based on “folding arguments”. These are interactive zero-knowledge protocols with logarithmic round complexity. The two most prominent examples of such protocols are Compressed  $\Sigma$ -protocols [3–5] and Bulletproofs [17, 19].

**Folding Technique.** The central object in all folding argument based *zero-knowledge* protocols is a sub-linear, interactive, logarithmic-round *non zero-knowledge* protocol to demonstrate that the prover has knowledge of a witness. The key idea used in the design of these logarithmic-round *non zero-knowledge* protocols is to enable the prover (using randomness from the verifier) to “fold” the witness in on itself, thereby reduce the size of the witness by half in each round. This step is repeated for a logarithmic number of rounds, until the witness is reduced to a constant size.

In order to build a sub-linear *zero-knowledge* protocol using the above *non zero-knowledge* protocol, most existing constructions rely on the same rough template—these constructions begin with a constant round “base” protocol containing a large final round message (*i.e.*, linear in the size of the original witness) that achieves zero-knowledge. Finally, instead of actually sending this large final round message, the prover uses the above (non zero-knowledge) recursive folding approach to prove knowledge of this large message over logarithmic rounds.

The key observation used here is that since the “base” protocol achieves zero-knowledge even if the large final round message is sent to the verifier in the clear, it suffices for the prover to use the above *non zero-knowledge* sublinear protocol to prove knowledge of this message.

**Folding Argument Based ZK-IPs are Stackable.** Most folding argument based zero-knowledge protocols including Compressed  $\Sigma$ -protocols and Bulletproofs fall into the category of sublinear-sized proofs, with verifier runtime roughly equivalent to the prover runtime. We observe that the folding arguments we study are *stackable* such that the prover’s entire last round message is recyclable.<sup>6</sup> To see this, note that if the last round message could instead be computed deterministically using the rest of the transcript, without access to the witness (which is the case for non-recyclable messages), then this last round message could also be computed by the verifier independently and there would be no need to send this message.<sup>7</sup> Specifically, this holds for the final round message in the “base” protocol in both Bulletproofs [17, 19] and Compressed  $\Sigma$ -protocols [3–5]. Because this last round message is recyclable, we observe that the entire folding argument—a proof of knowledge of a recyclable message—is itself recyclable and can be reused across clauses. We note, however, that the mere fact that these protocols are stackable doesn’t immediately imply that there are vast computational savings available when stacking folding arguments. Interestingly, we find that stacking Compressed  $\Sigma$ -protocols offers significant computational savings, while stacking Bulletproofs does not.

**Computational Savings via Stacking.** As discussed earlier, our hope to get computational savings when stacking sublinear zero-knowledge proof systems for disjunctions, stems from the observation that the simulator in such proofs is typically much faster than the prover algorithm. This is because, the verifier in most such protocols runs in sublinear time and since the job of the simulator is to essentially “fool” the verifier into accepting a simulated proof, the work required from a simulator is somewhat proportional to the work done by the verifier. As a result, being able to replace the prover algorithm with the simulator for all inactive clauses in the disjunction, can yield significant computational savings.

Folding argument based sublinear proof systems we consider, however, do not have a sublinear-time verifier. In fact, the work done by the verifier in these protocols is asymptotically equivalent to the work done by the prover. Hence, the overall simulator is not asymptotically more efficient than the prover algorithm. For computational savings, here we rely on our second observation about simulators: the simulator can often be split into two parts  $\mathcal{S}_{\text{RAND}}$  and  $\mathcal{S}_{\text{DET}}$ , where  $\mathcal{S}_{\text{RAND}}$  is responsible to simulating the statement independent part of the transcript, while  $\mathcal{S}_{\text{DET}}$  simulates messages that are dependent on the statement/relation. Since the messages simulated by  $\mathcal{S}_{\text{RAND}}$  are statement independent, the resulting messages can be reused/recycled in all the inactive clauses, while we must

<sup>6</sup> We formalize this claim in the full version [28].

<sup>7</sup> We do note, however, that some protocols include deterministic messages in the final round in order to minimize verifier computation.

compute the messages simulated by  $\mathcal{S}_{\text{DET}}$  separately for each clause. If  $\mathcal{S}_{\text{DET}}$  is significantly faster than  $\mathcal{S}_{\text{RAND}}$ , we can still hope to get significant concrete computational savings for the prover when stacking such protocols (even if there are no asymptotic savings). This is where the crucial difference between Bulletproofs and Compressed  $\Sigma$ -protocols appears.

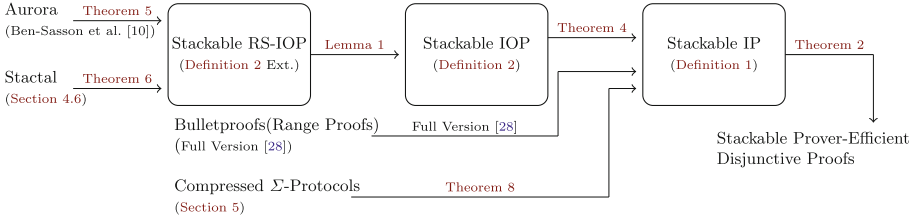
In Compressed  $\Sigma$ -protocols, the statement/relation-dependent verifier computation only consists of simple field operations, while the statement independent verification consists of expensive group multiexponentiations. As a result  $\mathcal{S}_{\text{DET}}$  is significantly more efficient than  $\mathcal{S}_{\text{RAND}}$ , yielding concrete computational savings for the prover upon stacking. Unfortunately, Bulletproofs lies on the other end of the spectrum, where the runtimes of  $\mathcal{S}_{\text{DET}}$  and  $\mathcal{S}_{\text{RAND}}$  are approximately the same (i.e. up to a small constant factor). This suggests to an interesting distinction between these two folding argument based protocols and motivates the design of sublinear-sized zero-knowledge protocols in which the verifier’s *statement-dependent* computation is much faster than the verifier’s *statement-independent* verifier computation—in other words, protocols that are more amenable to speed-stacking. We now give a brief technical overview of Compressed  $\Sigma$ -protocols and Bulletproofs to further highlight this distinction and demonstrate stackability.

**Compressed  $\Sigma$ -Protocol.** Compressed  $\Sigma$ -protocols [3–5] provide zero-knowledge interactive protocols for proving knowledge of openings of linear forms, i.e., proving that the output of a linear function  $f$  applied to a vector  $\mathbf{x}$  contained in a commitment  $P$  equals some publicly known value  $y$ . The “base” protocol in Compressed  $\Sigma$ -protocols, performs a randomized self-reduction, in which the problem is reduced to the task of proving a different (related) statement for the same relation in a *privacy-free* way. To prove this related statement, they provide a log-sized privacy-free argument.

We observe that the entire folding argument transcript can be reused during stacking (after making very minor modifications to the protocol), but not all of the computation can be reused. That is, the randomized simulator  $\mathcal{S}_{\text{RAND}}$  creates the folding argument transcript and then deterministic simulator  $\mathcal{S}_{\text{DET}}$  completes the transcript, but runs in time linear in the size of the vector  $\mathbf{x}$  ( $\mathcal{S}_{\text{DET}}$  recursively folds the linear form to facilitate the final check). However, we observe that the linear number of operations in  $\mathcal{S}_{\text{DET}}$  are all *field arithmetic*, and  $\mathcal{S}_{\text{DET}}$  contains only a single group exponentiation and a single group multiplication, with no multi-exponentiations. As a result, simulating each additional inactive clause remains significantly faster than the prover algorithm.

We note that we are able to handle disjunctions where each clause  $i \in [\ell]$  could have a different homomorphic linear function  $f_i$  and a different commitment  $P_i$ . This is a stronger notion of disjunctions than the ones considered in [4], which give proofs where either the homomorphism or commitment is fixed across a disjunction of multiple clauses.

**Bulletproofs.** The main task in the initial “base” protocol in Bulletproofs [19] is reduced to transforming any given relation into a *privacy-free* inner-product relation. This is followed by an efficient folding argument for  $\mathcal{R}_{\text{innerprod}}$ . This approach is used to achieve efficient zero-knowledge for range proofs and circuit



**Fig. 1.** A roadmap for the results in our paper. Several Theorems are contained in the full version of the paper.

satisfiability. Because the last message of the “base” protocol is recyclable, the folding argument transcript can be reused, and we find that only two of the messages in the “base” protocol are non-recyclable. However, simulating these two non-recyclable messages requires performing multi-exponentiations dependent on the relation function. As a result, any savings obtained from being able to recycle the entire *non zero-knowledge* sublinear-sized folding argument at the end across all inactive clauses are more-or-less eclipsed by the computation involved in individually simulating the above two non-recyclable messages for each inactive clause.

As such, stacking Bulletproofs for “true” disjunctions does not seem to offer considerable savings. We do note, however, one might consider set-membership for range proofs (i.e.  $\exists x_i \in \{x_1, \dots, x_\ell\}$  st.  $x_i \in \text{Range}$ ), where appealing to NP completeness is expensive. Because the statement dependent computation (that must be run separately for each clause) is remarkably inexpensive (involving only one group exponentiation and a constant number of group multiplications), applying the compiler in this case may be valuable. While set membership for range proofs is not particularly valuable, studying Bulletproofs illuminates fundamental differences between Compressed  $\Sigma$ -protocols and Bulletproofs, despite their superficial similarities. Moreover, this highlights the key parameters to keep in mind when stacking a protocol and points to new considerations when designing new—potentially stackable—zero-knowledge proof systems.

**Roadmap to Our Results.** We give an overview of how we reach our technical results in Fig. 1.

### 2.5 Notation

When discussing interactive protocols in this work, we will use both interactive Turing Machine notation, *ie.*  $\langle P, V \rangle(\mathbf{x})$ , and algorithmic notation, *ie.* the  $i^{\text{th}}$  message is computed with algorithm  $P_i$ . More formally, we assume that for zero-knowledge interactive proofs, the interaction  $\langle P, V \rangle(\mathbf{x})$  contains an ordered list of algorithms  $P_i$ , such that the prover computes their  $i^{\text{th}}$  message using  $P_i$ . We use  $\text{CC}(II)$  to denote the communication complexity of  $II$ , and let  $\text{Time}(II)$  denote the runtime of the algorithm  $II$ . Finally, we note that our work spans different lines of research that commonly leverage different notation for the same

concepts. Wherever possible we have made notation internally consistent, at the cost of being inconsistent with prior work.

For an NP relation  $\mathcal{R}$ , we denote the instance as  $\mathbf{x}$  and the witness as  $\mathbf{w}$ . Let the number of clauses in the disjunction  $\ell$  and the index of the active clause as  $\mathbf{a}$ . Where applicable, we use  $N$  to denote the relevant size of  $\mathbf{x}$ . We use multiplicative notation for groups and group operations.

### 3 Stacking Zero-Knowledge Interactive-Proofs

In this section, we extend the notion of “stackability” introduced in Goel et al. [26] to the multi-round setting proceed to give a generic compiler that can transform a stackable ZK-IP into a ZK-IP for disjunctive statements. We formally define Stackable ZK-IP in Sect. 3.1, present our stacking compiler in Sect. 3.2, and provide a heuristic mechanism for preparing ZK-IP protocols for stacking in the full version of our paper [28].

#### 3.1 Defining Stackable ZK-IP

Recently, Goel et al. [26] introduced the notion of “stackability” for  $\Sigma$ -protocols (i.e., three-round ZK-IPs), and showed most natural  $\Sigma$ -protocols are stackable. At the heart of their approach is the observation that the simulators for common  $\Sigma$ -protocols can be divided into two components: a randomized, statement independent part, which we will denote  $\mathcal{S}_{\text{RAND}}$ ,<sup>8</sup> and a deterministic, statement dependent part, which we denote  $\mathcal{S}_{\text{DET}}$ .

We extend their intuition to the multi-round setting. Intuitively, we require that each message in the protocol can be subdivided into two (potentially empty) parts: a *recyclable part* that can be reused across multiple statements, and a deterministically computable part. More formally, we assume that each prover message  $i$  of a ZK-IP is a concatenation of two parts— $m_{\text{RAND},i}$  and  $m_{\text{DET},i}$ . To satisfy stackability, we require that it is possible to generate the messages  $\{m_{\text{RAND},i}\}_{i \in [k]}$  using a randomized, statement independent algorithm  $\mathcal{S}_{\text{RAND}}$ . Additionally, we require that there exists a deterministic simulator  $\mathcal{S}_{\text{DET}}$  that can simulate the remaining parts of the messages  $\{m_{\text{DET},i}\}_{i \in [k]}$  such that the resulting transcript matches an “honest” execution of the protocol.

**Definition 1 (Stackable ZK-IP).** *Let  $\Pi$  be a ZK-IP consisting of  $k$  prover messages and  $k - 1$  verifier messages for a relation  $\mathcal{R}$ . For each  $i \in [k]$ , let  $m_i = (m_{\text{RAND},i}, m_{\text{DET},i})$  and let  $M_{\text{RAND}} = (m_{\text{RAND},i})_{i \in [k]}$  and  $M_{\text{DET}} = (m_{\text{DET},i})_{i \in [k]}$ . We say that  $\Pi$  is Stackable, if there exists a PPT simulator  $\mathcal{S}_{\text{RAND}}$  and a polynomial-time computable, well-behaved deterministic simulator  $\mathcal{S}_{\text{DET}}$ , such that for each  $C = (c_i)_{i \in [k-1]} \in \{0, 1\}^{\kappa k-1}$  and for all instance-witness pairs  $(\mathbf{x}, \mathbf{w})$  st.  $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$ , it holds that:*

<sup>8</sup> We depart from the notation introduced by Goel et al. [26], in which this first part is instead discussed as an efficiently samplable distribution, rather than a simulator. We note that these notions are clearly equivalent: the output of  $\mathcal{S}_{\text{RAND}}$  defines a distribution from which elements can be efficiently sampled (namely, by running  $\mathcal{S}_{\text{RAND}}$ ).

$$\left\{ (M, C) \mid r^P \stackrel{\$}{\leftarrow} \{0, 1\}^\lambda; \forall i \in [k], (m_{\text{RAND},i} \parallel m_{\text{DET},i}) \leftarrow \mathbf{P}_i(\mathbb{x}, \mathbb{w}, (c_j)_{j \in [i-1]}; r^P) \right\}$$

$$\approx$$

$$\left\{ ((m_{\text{RAND},i} \parallel m_{\text{DET},i})_{i \in [k]}, C) \mid M_{\text{RAND}} \leftarrow \mathcal{S}_{\text{RAND}}(1^\lambda, C); M_{\text{DET}} := \mathcal{S}_{\text{DET}}(\mathbb{x}, C, M_{\text{RAND}}) \right\}$$

The natural variants (perfect/statistical/computational) are defined depending on the class of distinguishers with respect to which indistinguishability holds.

### 3.2 Compiler for Stacking ZK-IPs

We now present our compiler that can transform any stackable ZK-IP into a ZK-IP for disjunctions. As discussed earlier, similar to Goel et al. [26], the main idea behind this construction is to honestly compute the transcript of the active clause and reuse its recyclable messages for all the inactive clauses.

Concretely, the prover starts by generating a  $(\text{ck}, \text{ek})$  pair for the index associated with the active clause. Subsequently, in each round it computes messages for the active clause honestly and commits to the non-recyclable messages along with a bunch of 0s for the inactive clause using the partially-binding vector commitment scheme and commitment key  $\text{ck}$ . It sends this commitment along with the honestly computed recyclable message to the verifier. In the last round, upon receiving all the challenge messages from the verifier, it simulates to “complete” the transcript of the inactive clauses and equivocates all of the previously computed commitments to a commitment of these messages and sends the associated commitment opening/randomness to the verifier. Based on the recyclable messages, the verifier also simulates the non-recyclable messages for each clause, and checks if they were honestly committed inside the commitment. It also checks if the resulting transcript for each clause is accepting.

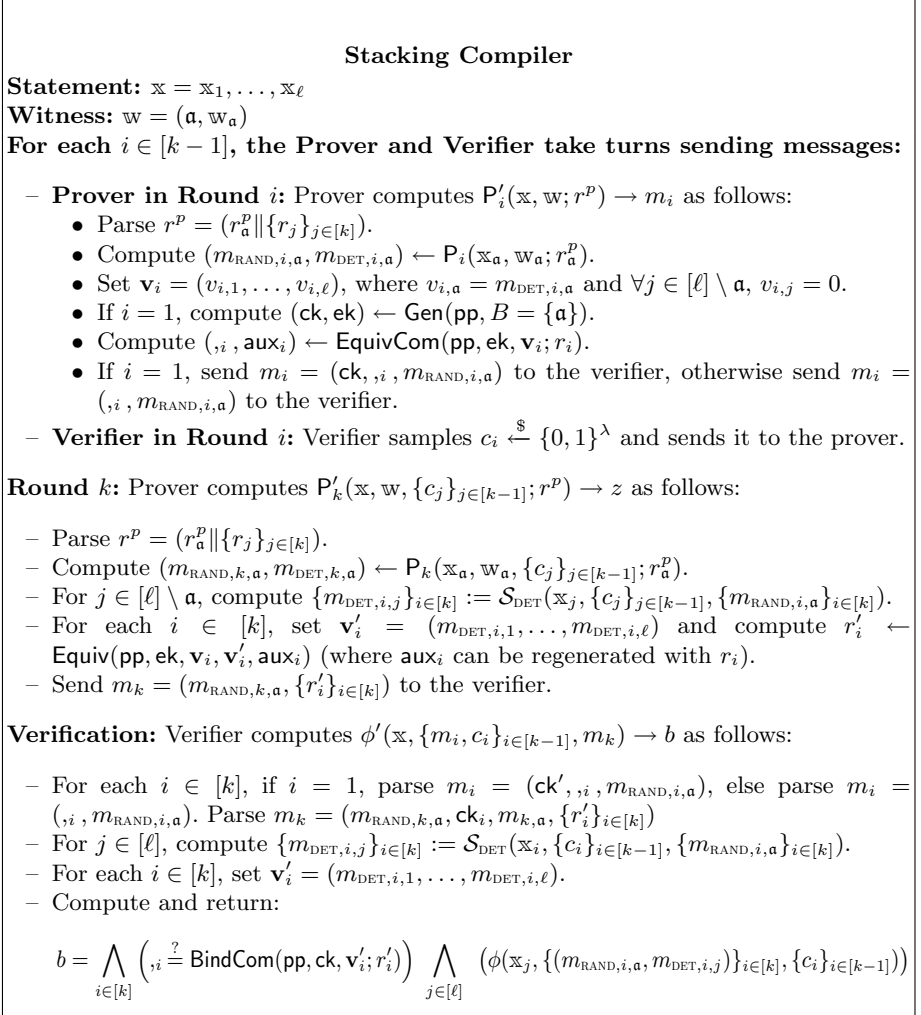
**Theorem 2.** *Let  $\Pi$  be a stackable ZK-IP (see Definition 1) consisting of  $k$  prover messages and  $k - 1$  verifier messages for the NP relation  $\mathcal{R} : \mathcal{X} \times \mathcal{W} \rightarrow \{0, 1\}$  and let  $(\text{Setup}, \text{Gen}, \text{EquivCom}, \text{Equiv}, \text{BindCom})$  be a 1-out-of- $\ell$  binding vector commitment scheme (as defined in [26]). For any  $\text{pp} \leftarrow \text{Setup}(1^\lambda)$ , the compiled protocol  $\Pi'$  described in Fig. 2 is a stackable ZK-IP for the relation  $\mathcal{R}' : \mathcal{X}^\ell \times ([\ell] \times \mathcal{W}) \rightarrow \{0, 1\}$ , where  $\mathcal{R}'((\mathbb{x}_1, \dots, \mathbb{x}_\ell), (\mathbf{a}, \mathbb{w})) := \mathcal{R}(\mathbb{x}_\mathbf{a}, \mathbb{w})$ .*

The proof for Theorem 2 can be found in the full version [28].

**Complexity Discussion.** Let  $\text{CC}(\Pi)$  be the communication complexity of  $\Pi$ . Then, the communication complexity of the  $\Pi'$  obtained from Theorem 2 is  $(\text{CC}(\Pi) + |\text{ck}| + |\text{com}| + |r'|)$ , where the sizes of  $\text{ck}, \text{com}$  and  $r'$  depend on the choice of partially-binding vector commitment scheme and are independent of  $\text{CC}(\Pi)$ . In the construction of partially-binding vector commitments from DLOG due to Goel et al. [26, Corollary 1],  $|\text{ck}|, |r'| = O_\lambda(\log \ell)$ , and  $|\text{com}| = O_\lambda(1)$ . Hence the communication cost of proving a disjunction of  $\ell$  clauses is  $O(\log \ell)$ .

Finding recyclable messages requires manual effort. We discuss intuition for finding these messages, along with an informal procedure, in the full-version of our paper [28].





**Fig. 2.** A compiler for stacking multiple instances of a stackable ZK-IP

## 4 Speed-Stacking Interactive Oracle Proofs

Interactive oracle proofs, originally proposed by [11, 45], form the basis of a widely-used framework for building succinct arguments. In this section we describe how to adapt this framework to build *stackable* succinct arguments.

We begin this section by recalling the preliminary definition of holographic IOPs (hIOPs), a generalization of IOPs introduced by [20] that allows for part of the input to be preprocessed, in Sect. 4.1. We then proceed to outline the technical machinery necessary to speed-stack two IOPs, Aurora IOP [10] Fractal hIOP [20]. Specifically, we use a series of compilers that speed-stacks these IOPs via several intermediary definitions. First, we define the notion of a *stackable*

(holographic) IOP in Sect. 4.3. Next, we describe how to transform a stackable IOP into a stackable (succinct) interactive argument, which can in-turn be speed-stacked using the compiler in Sect. 3. Finally, in Sect. 4.5, we describe our two constructions of stackable hIOPs, based on the Aurora IOP [10] and Fractal hIOP [20] constructions respectively.

### 4.1 Holographic IOPs

Holographic IOPs were originally defined in [20]. Here we describe special properties of holographic IOPs that we will make use of in this work; for a full definition of the model, see [28].

**Public Coins and Oblivious Queries.** In this work we will consider a certain subclass of IOPs: *public-coin* IOPs with *oblivious* queries. An IOP is public coin if each verifier message to the prover is a random string. This means that the verifier’s randomness  $C$  consists of its messages  $c_1, \dots, c_{k-1} \in \{0, 1\}^*$  and possibly additional randomness  $c_k \in \{0, 1\}^*$  used after the interaction (in particular, for choosing the query set). An IOP has *oblivious queries* if the verifier can be partitioned into a query algorithm  $V_Q$  and a decision algorithm  $V_D$  as follows.  $V_Q$  takes as input  $C$  (and nothing else) and outputs query sets  $(Q_1, \dots, Q_k)$ .  $V_D$  takes as input  $(\mathbb{x}, C, \Pi_1|_{Q_1}, \dots, \Pi_k|_{Q_k})$  and outputs a bit  $b$ .

**Zero-Knowledge.** A public-coin holographic IOP HOL has (perfect) special honest verifier zero-knowledge if there exists a probabilistic polynomial-time simulator  $\mathbf{S}$  such that for every  $(\mathbf{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$  the random variables  $\text{View}(\mathbf{P}(\mathbf{i}, \mathbb{x}, \mathbb{w}), \mathbf{V}^{\mathbf{I}(\mathbf{i})}(\mathbb{x}; C))$  and  $(C, \mathbf{S}(\mathbf{i}, \mathbb{x}, C, V_Q(C)))$  are identical, where:

- $C = (c_1, \dots, c_{k-1}, c_k)$  is the verifier’s (public) randomness, chosen uniformly at random, and
- $\text{View}(\mathbf{P}(\mathbf{i}, \mathbb{x}, \mathbb{w}), \mathbf{V}^{\mathbf{I}(\mathbf{i})}(\mathbb{x}; C))$  is the *view* of  $\mathbf{V}$  when interacting with  $\mathbf{P}$ , i.e., it is the random variable  $(C, \Pi_1|_{Q_1}, \dots, \Pi_k|_{Q_k})$ .

### 4.2 Reed–Solomon Encoded Holographic IOPs

*Reed–Solomon encoded IOPs* (RS-IOPs) were introduced in [10] and adapted to the holographic setting in [20, Section 4.1]. We refer the reader to [28] for a full definition of RS-IOPs; here we give an adapted definition of zero-knowledge for RS-IOPs that we will use later.

**Zero-Knowledge.** Honest-verifier zero-knowledge for RS-IOPs is trivial, since the honest RS-IOP verifier makes no queries, and so learns nothing from the interaction. Instead, we introduce a notion of *special semi-honest verifier* zero-knowledge (SSHVZK), which guarantees zero-knowledge against verifiers that behave honestly during the interaction, and then make a bounded number  $\mathbf{b}$  of arbitrary queries. Formally, an RS-IOP is SSHVZK with query bound  $\mathbf{b}$  if there exists a PPT simulator  $\mathbf{S}$  such that for every  $(\mathbf{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ , every large enough  $\ell \in \mathbb{N}$  and every function  $Q: \{0, 1\}^\ell \rightarrow \binom{L}{\mathbf{b}}$ , the random variables  $\text{View}_{Q(C)}(\mathbf{P}(\mathbf{i}, \mathbb{x}, \mathbb{w}), \mathbf{V}^{\mathbf{I}(\mathbf{i})}(\mathbb{x}))$  and  $(C, \mathbf{S}(\mathbf{i}, \mathbb{x}, C, Q(C)))$  are identical, where

- $C = (c_1, \dots, c_{k-1}, c^*)$ , chosen uniformly at random, is the verifier’s (public) randomness, (possibly) augmented to  $\ell$  bits with additional randomness  $c^*$ , and
- $\text{View}_{Q(C)}(\mathbf{P}(\mathfrak{i}, \mathfrak{x}, \mathfrak{w}), \mathbf{V}^{\mathbf{I}(\mathfrak{i})}(\mathfrak{x})) = (C, \Pi_1|_{Q(C)}, \dots, \Pi_k|_{Q(C)})$  is the view of the verifier in the protocol (which consists only of its own messages), augmented with the restriction of each prover message to the set  $Q(C) \subseteq L$ .

### 4.3 Defining a Stackable IOP and Stackable RS-IOP

In this section we give definitions for a *stackable RS-IOP* and a *stackable IOP*, before showing how to compile from the former to the later in Sect. 4.4. Looking ahead, we will give modifications of Aurora IOP [10] and Fractal hIOP [20] that are stackable RS-IOPs. The two definitions are defined in largely the same way; the differences are analogous to the differences between an RS-IOP and IOP. As such, we only explicitly give the definition of a stackable IOP, as the generalization is trivial.

Recall that the simulator for a ZKIOP is required to sample answers for exactly the points that the honest verifier queries in each round; these points are provided to the simulator as a vector  $\mathbf{Q} = (Q_1, \dots, Q_k)$ , where  $Q_i$  is the set of points that the verifier queries in round  $i$ . Hence we can write the simulator’s output as a sequence of functions  $\Pi_i^* : Q_i \rightarrow \Sigma$ , where  $\Sigma$  is the alphabet of the IOP. Given this template, stackability for IOPs is defined similarly to stackability for IPs (Definition 1), as follows.

**Definition 2 (Stackable hIOP).** *We say that an  $k$ -round holographic IOP  $\text{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$  is stackable if there exists a subset of “recyclable” rounds  $R_{\text{rec}} \subseteq [k]$  and a pair of algorithms  $(\mathcal{S}_{\text{RAND}}, \mathcal{S}_{\text{DET}})$  where  $\mathcal{S}_{\text{DET}}$  is deterministic, such that for all  $(\mathfrak{i}, \mathfrak{x}, \mathfrak{w}) \in \mathcal{R}$ , the following algorithm is a special honest-verifier zero-knowledge simulator for  $\text{HOL}$ :*

$\mathbf{S}(\mathfrak{i}, \mathfrak{x}, C, \mathbf{Q})$ :

1. sample  $(\Pi_i^* : Q_i \rightarrow \Sigma)_{i \in R_{\text{rec}}} \stackrel{\$}{\leftarrow} \mathcal{S}_{\text{RAND}}(C, \mathbf{Q})$ ;
2. compute  $(\Pi_i^* : Q_i \rightarrow \Sigma)_{i \in [k] \setminus R_{\text{rec}}} := \mathcal{S}_{\text{DET}}^{\mathbf{I}(\mathfrak{i})}(\mathfrak{x}, (\Pi_i^*)_{i \in R_{\text{rec}}}, C, \mathbf{Q})$ ;
3. output  $(\Pi_i^*)_{i \in [k]}$ ;

and for all  $\lambda \in \mathbb{N}$  and  $(\mathfrak{i}', \mathfrak{x}', \mathfrak{w}')$  (whether in  $\mathcal{R}$  or not),  $\mathbf{S}(1^\lambda, \mathfrak{i}', \mathfrak{x}')$  outputs an accepting view with certainty.

The definition extends in the natural way to Reed–Solomon encoded IOPs (RS-IOPs), except that we require that  $\mathbf{S}$  be an SSHVZK simulator.

### 4.4 Compiling RS-IOP to Stackable IP via Stackable IOP

In this section we show how to “compile” a stackable RS-IOP into a stackable IOP, and a stackable IOP into a stackable IP using a key-value commitment schemes. In the full version [28], we give a formal definition for the key-value commitment schemes that we require. In this section we provide both compilers (in Lemma 1 and Theorem 4 respectively).

**Hiding Key-Value Commitments.** Key-value commitments, described by Boneh, Bünz and Fisch [16] and Agrawal and Raghuraman [2] primarily in blockchain-related applications are a generalization of vector commitments: allowing the committer to efficiently commit to a (potentially) exponentially large but sparse vector in time that is polynomial in the security parameter and the number of entries in the sparse vector. Unlike the primary motivation for these works, we are not concerned with updateability of the map; however, we additionally require the commitments to hide the unopened entries. The exact definition and constructions can be found in the full-version of our paper [28].

**Compiling RS-IOP to Stackable IOP.** We now show that, by slightly tweaking the RS-IOP to IOP transformation presented in [10, Section 8.1], we can preserve stackability. The compiler of [10, Section 8.1] converts an RS-IOP into an IOP using a (IOP) proximity test for Reed-Solomon codes [8, 13] (also called a Low-Degree Test (LDT)). Since the concrete cost of the proximity test is large, by exploiting the linearity of the code, all the oracles are combined using a random linear combination into a *single* claimed codeword; rather than repeating the proximity for every individual oracle. This incurs a soundness-error of  $1/|\mathbb{F}|$ <sup>9</sup>. This works for codewords in the *same code*, to account for multiple RS codes of different rate note that component-wise products of Reed-Solomon codes is a Reed-Solomon code, i.e. for a fixed  $C_1 \in \text{RS}[L, d_1]$ :  $C_1 \circ C_2 \in \text{RS}[L, d_1 + d_2] \iff C_2 \in \text{RS}[L, d_2]$ . This allows homogenizing all the rates: for the verifier to query  $(C_1 \circ C_2)(i)$  simply query  $C_2(i)$  and compute  $C_1(i) \cdot C_2(i)$ , hence we can assume that the rate of all codewords is the same. Note that  $C_1$  can be an arbitrary codeword, in particular it can be chosen such that computing  $C_1(i)$  is very efficient. Lastly, since the proximity test is not zero-knowledge the prover samples a random codeword which is added to the linear combination: such that the distribution of the codeword on which the proximity test is run is uniform. In summary, the verifier samples  $\mathbf{z} \in \mathbb{F}^k$  and the proximity test is run on the oracle:

$$q = \mathbf{z}^T \Pi + r$$

for codewords  $\Pi \in \text{RS}[L, d]^k$  and  $r \in \text{RS}[L, d]$ . Note that  $q(i)$  can be accessed by simply querying  $\Pi$  and  $r$  at  $i$ , hence in [10, Protocol 8.6] there is no need for the prover to send the oracle  $q$  explicitly<sup>10</sup>, however we need this to efficiently stack.

**Lemma 1 (From Stackable RS-IOP to Stackable IOP).** *There is a transformation (an adaptation of [10, Protocol 8.6]) which composes a stackable RS-IOP and any IOPP for the Reed–Solomon code (i.e., a low-degree test) to produce a stackable IOP for the same relation. Moreover, the cost of  $\mathcal{S}_{\text{DET}}$  for the resulting IOP is the same as the cost of  $\mathcal{S}_{\text{DET}}$  for the RS-IOP. The construction follows easily from the discussion above, see full version [28] for details.*

<sup>9</sup> For fields where  $1/|\mathbb{F}|$  is not negligible, parallel repetition is used: requiring repetitions of the proximity test as well.

<sup>10</sup> Which would also require an additional proximity test between  $q$  and  $\mathbf{z}^T \Pi + r$ .

**Compiling Stackable IOP to Stackable IP.** Next we show how to compile stackable IOPs into stackable interactive arguments using hiding key-value commitments. The construction is an adaptation of the natural construction of a succinct argument from an IOP using vector commitments; the security and efficiency guarantees of hiding key-value commitments are necessary to preserve stackability and stacking efficiency of the underlying IOP.

**Construction 3** (Stackable IOP to Stackable IP Compiler). Assume wlog. That the (public coin)  $\mathbf{V}^{\mathbf{I}^{(i)}}(\mathbf{x})$  only makes queries to the oracles after the  $k$ 'th round and transform an  $k$ -round holographic IOP into a  $k + 1$  stackable IP follows: In round  $i$ , when  $\mathbf{P}(\mathbf{i}, \mathbf{x}, \mathbf{w})$  outputs  $\Pi_i$ , compute the commitment to the oracle:

$$(C_i, \mathfrak{o}_i) \leftarrow \text{KV.Com}(\text{pp}, \{(j, \Pi_i(j))\}_{j \in [|\Pi_i|]})$$

And sends  $C_i$  to  $\mathbf{V}$ . After round  $k$ ,  $\mathbf{V}$  outputs the set of queries  $\mathbf{Q} = \{Q_i\}_{i \in [k]}$  to each oracle  $\Pi_i$ . The prover  $\mathbf{P}$  responds by opening the key-value commitments at the requested positions: for all  $i \in [k]$  defining  $\mathcal{M}_i = \{(j, \Pi_i(j))\}_{j \in Q_i}$ , followed by sending  $\mathcal{M}_i$  and  $\mathfrak{o}_i \leftarrow \text{KV.Open}(\text{pp}, \mathfrak{o}_i, \mathcal{M})$  to  $\mathbf{V}$ . The transformation above is essentially the one by Ben-Sasson et al. [11, Section 6] (from IOPs to IPs) but replacing Merkle trees with the related notion of a key-value commitment.

**Theorem 4 (Correctness of Construction 3).** Given a key-value commitment scheme: a stackable holographic IOP  $\text{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$  can be compiled into an efficient stackable interactive argument  $(\mathbf{P}, \mathbf{V})$ . Furthermore, the running time of the compiled  $\mathcal{S}_{\text{DET}}$  is that of  $\mathcal{S}_{\text{DET}}^{\mathbf{I}^{(i)}}$  from the IOP, plus that of computing  $(C_1, \dots, C_k)$ , which is  $O(\sum_i |\Pi_i^*| \cdot \text{poly}(\lambda, \log(|\Pi_i|)))$  (where  $|\Pi_i|$  is the length of the  $i$ -th oracle in the real execution). See full version [28] for the proof.

### 4.5 Stackable RS-IOPs

In this section we show that two key IOP protocols from the literature, Aurora [10] and Fractal [20] can be made stackable. These protocols are proof systems for the R1CS relation, defined formally below.

**Definition 3.** Rank-one constraint satisfiability (R1CS) is an indexed NP relation consisting of all index-instance-witness tuples  $((\mathbb{F}, A, B, C), x, w)$  for  $A, B, C \in \mathbb{F}^{n \times n}$ ,  $x \in \mathbb{F}^k$ ,  $w \in \mathbb{F}^{n-k}$ , such that for  $z = (x \| w)$ ,  $Az \circ Bz = Cz$ , where  $\circ$  is the element-wise product.

Before proceeding to discuss how to make these protocols stackable, we provide a brief overview of the Aurora and Fractal RS-IOPs. These descriptions are not comprehensive, but rather aim to give context for the stackable variants presented later. Both protocols start from the same basic template:

1. On input  $((\mathbb{F}, A, B, C), x, w)$ , the prover sends to the verifier a (Reed-Solomon) encoding  $f_w$  of  $w$ , from which the verifier can deterministically compute an encoding  $f_z$  of  $z = (x \| w)$ . The prover also computes vectors  $Az, Bz, Cz$  and sends their corresponding encodings  $f_A, f_B, f_C$  to the verifier.

2. For each  $M \in \{A, B, C\}$ , the prover and verifier engage in the “lincheck” protocol to show that  $f_M$  is an encoding of  $Mz$ . This involves one or two rounds of interaction for Aurora and Fractal respectively, after which the verifier will output some rational constraints.
3. Lastly the verifier outputs the constraint “ $f_A(i) \cdot f_B(i) - f_C(i) = 0$  for all  $i \in [n]$ ”.

To achieve zero-knowledge, the encodings  $f_w, f_A, f_B, f_C$  are *randomized* so that any “view” consisting of  $b$  locations in the encoding is distributed as a uniformly random vector in  $\mathbb{F}^b$ ; hence the messages sent in Step 1 are recyclable. Because the prover does not send any information in Step 3, it is not relevant for zero-knowledge or stackability. As such, we need only focus on Step 2. Indeed, the difference between Aurora and Fractal lies in this step: Aurora’s lincheck has verification time linear in the number of nonzero entries in  $A, B, C$ , whereas Fractal’s lincheck is exponentially faster after preprocessing. As a result, they behave quite differently when stacked.

**Aurora is Stackable.** We show that a small modification to Aurora yields a stackable RS-IOP. We first outline the lincheck protocol used in Aurora. Both the prover and verifier take as input a matrix  $M$ , and have access to Reed–Solomon codewords  $f, f_M$ , which purportedly satisfy the relation  $f_M|_H = Mf|_H$  for specified  $H \subseteq \mathbb{F}$ . For  $\alpha \in \mathbb{F}$ , denote by  $\mathbf{u}_\alpha$  the vector  $(1, \alpha, \alpha^2, \dots, \alpha^{|H|-1}) \in \mathbb{F}^H$ .

1. The verifier sends a challenge point  $\alpha \in \mathbb{F}$ .
2. The prover and verifier both compute the vector  $\mathbf{u}_\alpha M \in \mathbb{F}^H$  along with its low-degree extension  $\hat{g}$ .
3. The prover and verifier then engage in the zero-knowledge sumcheck protocol to show that

$$\langle \mathbf{u}_\alpha M, f|_H \rangle - \langle \mathbf{u}_\alpha, f_M|_H \rangle = \sum_{a \in H} \hat{u}_{\alpha, M}(a) f(a) - \hat{g}(a) f_M(a) = 0 .$$

This protocol is complete because if  $f_M|_H = Mf|_H$  then for all vectors  $u$ ,  $\langle u, f_M|_H \rangle = \langle u, Mf|_H \rangle = \langle uM, f|_H \rangle$ . For soundness, observe that if  $f_M|_H \neq Mf|_H$  then  $\langle \mathbf{u}_\alpha M, f|_H \rangle - \langle \mathbf{u}_\alpha, f_M|_H \rangle$  is a nonzero low-degree polynomial in  $\alpha$ ; soundness follows by elementary algebra and the soundness of the zero-knowledge sumcheck protocol.

Observe that the only prover-to-verifier communication in this lincheck protocol is within Step 3; specifically, in the execution of zero-knowledge sumcheck. We now recall (and slightly modify) the zero-knowledge sumcheck protocol, which relies on the following lemma.

**Lemma 2 (By Ben-Sasson et al. [10]).** Let  $H$  be a coset of an additive or multiplicative subgroup of  $\mathbb{F}$ . Then there is a polynomial  $\Sigma_{H, Y}(X)$ , which can be evaluated in time  $\text{polylog}(|H|)$ , such that the following holds: let  $\hat{f} \in \mathbb{F}[X]$  be such that  $\text{deg}(\hat{f}) < |H|$ . Then  $\sum_{\alpha \in H} \hat{f}(\alpha) = \sigma$  if and only if there exists  $\hat{g}$  with  $\text{deg}(\hat{g}) < |H| - 1$  such that  $\hat{f}(X) \equiv \Sigma_{H, \sigma}(\hat{g}(X))$ .

The protocol proceeds as follows: The prover and verifier have access to a summand codeword  $f$  of degree  $d$ , which purportedly satisfies  $\sum_{a \in H} \hat{f}(a) = 0$ .

1. The prover chooses a random polynomial  $\hat{r}$  of degree  $d$ , computes  $\zeta = \sum_{a \in H} \hat{r}(a)$ , and sends  $r, \zeta$  to the verifier.
2. The verifier sends a challenge  $\beta$ .
3. The prover divides  $\hat{q} := \hat{r} + \beta \hat{f}$  by  $v_H$  to obtain  $\hat{g}, \hat{h}$  satisfying the identity  $\hat{q} \equiv \Sigma_{H, \zeta}(\hat{g}) + \hat{h} \cdot v_H$  with  $\deg(\hat{g}) < |H| - 1$ , and sends  $h$  to the verifier.
4. The verifier outputs the rational constraint “ $\deg(\hat{e}) < |H| - 1$ ”, where  $\hat{e} := \Sigma_{H, \zeta}^{-1}(\hat{q} - \hat{h} \cdot v_H)$ .

The zero-knowledge simulator given by [10] operates by first choosing a random polynomial  $\hat{q}$ , and sending  $\zeta = \sum_{a \in H} \hat{q}(a)$  in the first round.  $\hat{g}, \hat{h}$  are obtained from this  $\hat{q}$  in the same way as the honest prover. Queries to  $r$  are answered using  $q - \beta f$ .

We are now ready to show that the above protocol is stackable, after a small modification.

**Theorem 5.** *The Aurora zero-knowledge RS-IOP for R1CS [10, Protocol 7.5] is stackable (after a small modification) with  $\mathcal{S}_{\text{DET}}$  running in time  $O(\|A\| + \|B\| + \|C\| + n \log^2 \mathbf{b} \log \log \mathbf{b})$  (measured in field operations).*

*Proof.* The only modification necessary is to the zero-knowledge sumcheck protocol. Specifically, in Step 3, the prover will also send  $g$ ; this is purely for the purposes of simulation and does not affect soundness.

Note that in a real execution,  $g, h$  are (marginally) uniformly random codewords, and so can be generated by  $\mathcal{S}_{\text{RAND}}$  (i.e., they are recyclable). Hence the only oracle in the protocol that is *not* recyclable is  $r$ . The inclusion of  $g$  in the protocol allows  $\mathcal{S}_{\text{DET}}$  to compute  $r$  as  $\Sigma_{H, \zeta}(g) + h \cdot v_H - \beta f$ .

As a result, the time complexity of  $\mathcal{S}_{\text{DET}}$  is dominated by the evaluation of  $f$  at  $\mathbf{b}$  points. This requires computing  $rA, rB, rC \in \mathbb{F}^n$  for some  $r \in \mathbb{F}^m$ , which takes  $O(\|A\| + \|B\| + \|C\|)$  field operations, and evaluating the low-degree extensions of these vectors at  $\mathbf{b}$  points, which takes  $O(n \log^2 \mathbf{b} \log \log \mathbf{b})$  operations using the algorithm of [18].

## 4.6 Stactal

Next we describe “stackable Fractal”, or Stactal, a variant of the Fractal protocol [20] which can be efficiently stacked. The verifier in Fractal runs in time quasi-linear in the length of the input vector  $x$  and *polylogarithmic* in the dimensions of  $A, B, C$ . This is achieved via a sparse holographic encoding of  $A, B, C$  using the Reed–Solomon code.

First, we discuss why directly stacking Fractal leads to an inefficient protocol. Recall that in the stacked protocol, the prover and verifier run the instance-dependent part of the simulator  $\mathcal{S}_{\text{DET}}$  on each clause  $j \in [\ell]$ . Therefore, to achieve the desired computational savings for the prover while maintaining the complexity of the verifier, we want  $\mathcal{S}_{\text{DET}}$  to run in polylogarithmic time. Unfortunately,

this is not possible for the original Fractal protocol (in the true disjunction setting), as we explain next.

The verifier’s running time in the Aurora protocol is dominated by the lincheck subprotocol: specifically, the cost of evaluating, for each input matrix  $M \in \{A, B, C\}$ , the low-degree extension  $\hat{u}_{\alpha, M}$  of the vector  $\mathbf{u}_{\alpha}M$ . To eliminate this cost, Fractal replaces Aurora’s lincheck protocol with a *holographic* variant. In particular, [20] shows that, given an appropriate encoding of the input matrices, there is a protocol that allows the verifier to check an evaluation of this low-degree extension in time  $\text{polylog}(\|M\|)$ .

Since the verifier cannot compute this evaluation itself, the prover sends  $\hat{u}_{\alpha, M}(\beta)$  for the desired evaluation point  $\beta$ . In the standard setting of zero-knowledge, since the input matrices are *public*, this is not a problem: the simulator can simply compute this evaluation as the honest prover would, in time linear in  $\|M\|$ . In the stacking setting, however, this computation would be part of  $\mathcal{S}_{\text{DET}}$ , more than negating the computation savings obtained via holography.

Worse, it is not possible to simply design a better simulator: for most choices of  $\alpha, \beta$ ,  $\hat{u}_{\alpha, M}(\beta)$  depends on every nonzero entry of  $M$ . Thus  $\mathcal{S}_{\text{DET}}$  must run in at least linear time. To resolve this, we must instead significantly modify the Fractal protocol. In more detail, we allow the prover to “pad” the input matrices with randomness, in a way that does not affect the satisfiability of the statement, so that  $\hat{u}_{\alpha, M}(\beta)$  becomes uniformly random. The simulator for this protocol runs in time  $\text{polylog}(\|M\|)$  and makes a small number of queries to the encoding of  $M$ . We prove the following theorem in the full version of the paper [28]:

**Theorem 6 (Stactal).** *The protocol obtained from Fractal by replacing the holographic lincheck protocol with a stackable holographic lincheck (as described in the full version of our paper [28]) is stackable, with  $\mathcal{S}_{\text{DET}}$  running in time  $O(b \cdot (|x| + \text{polylog}(\|A\| + \|B\| + \|C\|)))$  (measured in field operations).*

## 5 Speed-Stacking Compressed $\Sigma$ -Protocols

We now turn our attention to stacking sublinear proofs based on folding arguments. “Folding arguments” refers to a class of proof systems that relies on algebraic structure and interaction to iteratively reduce the size of (or “fold”) the statement of interest. The two most notable instantiations of this class are Bulletproofs [19], which give a folding argument for inner products, and Compressed  $\Sigma$ -Protocols [3–5], which give folding arguments for linear forms. In this section, we show how to stack Compressed  $\Sigma$ -Protocols and demonstrate the computational savings that our techniques offer when applied to them. In the full-version [28], we demonstrate how to stack Bulletproofs, which as discussed earlier are less amenable to computational savings from our stacking approach.

Compressed  $\Sigma$ -protocols were proposed in a series of recent works by Attema, Cramer, Fehr and Kohl [3–5]. In this section, we focus on the specific instantiation of this approach proposed by Attema, Cramer, and Fehr [4], as it has a clean presentation.



**Notation.** We slightly modify some of the notation presented by Attema, Cramer, and Fehr in [4] for clarity of presentation, but endeavor to make it sufficiently consistent that an interested reader can easily refer back to their work for additional details. Let  $\mathbb{G}$  be a cyclic group of prime order  $p$ . Let  $f$  be a homomorphism from (additive)  $\mathbb{Z}_q^N$  to some group  $\mathbb{G}_T$ .<sup>11</sup> We denote the set of such homomorphisms as  $\mathcal{L}^N$ .

Let  $\mathbf{g}_i = (g_{i,1}, g_{i,2}, \dots, g_{i,N})$  be vectors of generators in  $\mathbb{G}$ , where the size of the vector will either be stated explicitly or, when clear from context, left implicit. All other lower-case letters, *e.g.*  $c, a$ , refer to elements in  $\mathbb{Z}_q$ , and bold lower-case letters, *e.g.*  $\mathbf{x}_i, \mathbf{z}$  refer to vector of elements in  $\mathbb{Z}_q$ . Let  $\mathbf{x} = \{x_1, \dots, x_M\} \in \mathbb{Z}_q^N$ , and  $f : \mathbb{Z}_q^N \rightarrow \mathbb{G}_T$ . We denote  $\mathbf{x}_L = \{x_1, \dots, x_{N/2}\}$  and  $\mathbf{x}_R = \{x_{N/2+1}, \dots, x_N\}$ . We denote  $f_L : \mathbb{Z}_q^{N/2} \rightarrow \mathbb{G}_T$  as the function  $f(\mathbf{x}_L, 0, \dots, 0)$  and  $f_R : \mathbb{Z}_q^{N/2} \rightarrow \mathbb{G}_T$  as the function  $f(0, \dots, 0, \mathbf{x}_R)$ . Upper-case letters refer to elements of  $\mathbb{G}$ . For a vector  $\mathbf{g}_i$  of length  $N$ , we denote the first  $N/2$  elements of  $\mathbf{g}_i$  as  $\mathbf{g}_{iL}$  and the remaining  $N/2$  elements of  $\mathbf{g}_i$  as  $\mathbf{g}_{iR}$ . We denote the element-wise group operation of two vectors of group elements as  $\mathbf{g} * \mathbf{g}' = (g_1 g'_1, \dots, g_N g'_N)$ , where  $N$  is an arbitrary size parameter. Finally we denote multi-exponentiation by  $\mathbf{g}^{\mathbf{x}} = \prod_i g_i^{x_i}$ .

**Compressed  $\Sigma$ -Protocols.** Attema et al. [4] consider the relation  $\mathcal{R}_{\text{compressed}} = \{(\mathbf{g} \in \mathbb{G}^N, P \in \mathbb{G}, y \in \mathbb{G}_T, f \in \mathcal{L}^N; \mathbf{x} \in \mathbb{Z}_q^N) : P = \mathbf{g}^{\mathbf{x}}, y = f(\mathbf{x})\}$ , where  $\mathbf{x}$  is a vector of length  $N$  and  $f$  is a homomorphism from  $\mathbb{Z}_q^N$  to  $\mathbb{G}_T$ . Intuitively, their protocol is a “standard” (Schnorr-type)  $\Sigma$ -protocol, where the prover computes  $\mathbf{r} \stackrel{\$}{\leftarrow} \mathbb{Z}_q^N, T = \mathbf{g}^{\mathbf{r}}$  and  $t = f(\mathbf{r})$  and sends  $t, T$  to the verifier. Upon receiving a challenge  $c$ , it computes and sends  $\mathbf{z} = c\mathbf{x} + \mathbf{r}$  to the verifier. The verifier then verifies if:  $\mathbf{g}^{\mathbf{z}} \stackrel{?}{=} TP^c$  and  $f(\mathbf{z}) \stackrel{?}{=} cy + t$  (later in this section, we will denote the value  $TP^c$  as  $Q$ ). Note that, the third round message  $\mathbf{z} \in \mathbb{Z}_q^N$  that the prover sends in this protocol contains  $O(N)$  elements, which is undesirable.

To compress the communication complexity of this last round message, this line of work makes the observation that the message  $\mathbf{z}$  is itself a trivial proof of knowledge for an instance of  $\mathcal{R}_{\text{compressed}}$ . Specifically,

$$\{(\mathbf{g} \in \mathbb{G}^N, TP^c \in \mathbb{G}, cy + t \in \mathbb{G}_T, f \in \mathcal{L}^N; \mathbf{z} \in \mathbb{Z}_q^N) : P = \mathbf{g}^{\mathbf{z}}, y = f(\mathbf{z})\}.$$

Importantly, however, sending  $\mathbf{z}$  reveals nothing about  $\mathbf{x}$ . As such, for reducing the communication complexity of the base protocol, it suffices to design a proof of knowledge for  $\mathcal{R}_{\text{compressed}}$  that need not be *zero-knowledge*. The various versions of Compressed  $\Sigma$ -Protocols design slightly different variants of this compressive “folding” proof of knowledge. In this work, we focus on the one presented in [4].

**Folding Argument.** They start by enabling the prover and the verifier to split the statement in half and fold it in on itself, resulting in a transcript that is

<sup>11</sup> Although  $\mathbb{G}_T$  is often used to indicate a target group in a pairing, in this context it simply refers to the target group of the homomorphism; there are no pairings here. Additionally, we encourage the reader to think of  $\mathbb{G}$  simply as  $\mathbb{Z}_q$ , as this is the clear motivation for the proof system.

half the size. This is done as follows: the verifier generates a random challenge  $c \in \mathbb{Z}_q$ , and the task of proving the original instance is reduced to the problem of proving another instance of  $\mathcal{R}_{\text{compressed}}$  for a new linear form  $f' = cf_L + f_R$  with bases  $\mathbf{g}' = \mathbf{g}_L^c * \mathbf{g}_R$ . Note the dimension of each of these is half the dimension of the original. All that remains now is to generate a new commitment  $P'$  and find a new target value  $y'$  for this reduced-dimension instance. The prover and verifier compute this as follows:

- (1) Before  $c$  is sent by the verifier, the prover computes  $A = \mathbf{g}_R^{\mathbf{x}_L}, a = f_R(\mathbf{x}_L), B = \mathbf{g}_L^{\mathbf{x}_R}, b = f_L(\mathbf{x}_R)$  and sends  $(A, a, B, b)$  to the verifier.
- (2) The verifier then samples and sends  $c$ .
- (3) The prover and verifier compute  $P' = AP^c B^{c^2}$  and  $y' = a + cy + c^2b$ .

The new instance is now of the form:

$$\left\{ (\mathbf{g}' \in \mathbb{G}^{N/2}, AP^c B^{c^2} \in \mathbb{G}, y' \in \mathbb{G}_T, f' \in \mathcal{L}^{N/2}; \mathbf{x}' \in \mathbb{Z}_q^{N/2}) : AP^c B^{c^2} = \mathbf{g}'^{\mathbf{x}'}, y' = f'(\mathbf{x}') \right\}$$

Note that a trivial proof of knowledge for this new instance is just  $\mathbf{x}' = \mathbf{x}_L + c\mathbf{x}_R$ , which is already half the length of the initial  $\mathbf{x}$ . The same process can be repeated again for computing a proof of knowledge of  $\mathbf{x}'$ , to further reduce the communication complexity. This process is recursively applied until the final trivial witness is of a constant size.

We note that Attema et al. have demonstrated how to use their protocol(s) to prove generic circuit satisfiability, by arithmetizing the circuit into a compatible format. We focus on the simpler base case where the prover only wishes to prove a linear form, and discuss the generalization in the full version of our paper [28]. We now state the main Theorem from [4].

**Theorem 7** ([4]). Let  $N > 2$ . There exists a  $(2\mu+3)$ -move protocol  $\Pi_{\text{compressed}}$  for relation  $\mathcal{R}_{\text{compressed}}$ , where  $\mu = \lceil \log_2(N) \rceil - 2$ . It is a perfectly complete, special honest-verifier zero-knowledge and unconditionally  $(2,3,3, \dots, 3)$ -special sound.

### 5.1 Compressed $\Sigma$ -Protocols are Stackable

We consider statements of the form:  $\mathcal{R}_{\text{dis-compressed}} = \{(\mathbf{g} \in \mathbb{G}^N, \{P_i \in \mathbb{G}, y_i \in \mathbb{G}_T^i, f_i\}_{i \in [\ell]}; \mathbf{a} \in [\ell], \mathbf{x}_a \in \mathbb{Z}_q^N) : P_a = \mathbf{g}^{\mathbf{x}_a}, y_a = f_a(\mathbf{x}_a)\}$ . Notice that this statement allows for different homomorphisms and commitments for each clause  $i \in [\ell]$ . This is a stronger notion of disjunctions than considered in [4], which give proofs where either the homomorphism or commitment is fixed across a disjunction of multiple clauses. Our goal in stacking will be concrete speed; specifically, we aim to minimize the number of expensive group operations and multi-exponentiations the prover is required to do for each clause.

**Intuition.** A first order intuition for speed-stacking Compressed  $\Sigma$ -Protocols is as discussed in the technical overview: first stack the communication inefficient base protocol, and then apply the recursive folding “after” stacking the protocols together. The base  $\Sigma$  protocol in Compressed  $\Sigma$ -Protocols can trivially be stacked using the stacking compiler given from Goel et al. [26], reusing

the entirety of  $\mathbf{z}$  as a recyclable message and allowing  $t, T$  to be deterministically recomputed. As such, it is natural to expect that this multi-round protocol should contain all recyclable messages besides  $t, T$ , and indeed it does.

We note, however, that Attema et al.’s choice of compression mechanism requires a more careful analysis of this stacking approach. Not all the messages in the tail are recyclable. Observe that the messages in the tail are of the form  $A_i, B_i, a_i, b_i$ . While  $A_i, B_i$  are clearly recyclable,  $a_i, b_i$  are outputs of some combination of parts of the linear form  $f$  and hence depend on  $f$ . Moreover, the *computation* required to verify the tail is also not reusable. Specifically, the linear form  $f$  is itself incorporated into the compression mechanism, and  $f$  is never blinded, *i.e.* the computations relying on  $f$  cannot be “recycled” (to slightly abuse our terminology). As such, directly stacking the protocol will run into both efficiency problems and difficulty in proving zero-knowledge (*i.e.*, in ensuring that the index of the active branch remains hidden).

We propose two minor modifications to this protocol to maximize stacking:

- (1) **Sending  $Q_1$ :** In the original protocol described in [4], the prover and verifier independently compute the value  $Q_1$  (*i.e.*,  $TP^c$  from the base protocol). The first modification that we propose is to have the prover send  $Q_1$  during the first folding. This modification is simply for *efficiency* reasons (and therefore does not impact soundness or zero-knowledge) as  $Q_1$  can be deterministically computed by the verifier and the deterministic simulator. However, computing  $Q_1$  directly from the transcript (and, looking ahead, the recyclable messages) for simulating other messages is *expensive* — involving many exponentiations — and therefore we would like to avoid computing it as part of our deterministic simulator  $\mathcal{S}_{\text{DET}}$ . This modification is similar to the one used to make Aurora efficiently stackable in the previous Section.
- (2) **Randomizing  $a_i$  and  $b_i$ :** In each round  $i$  of the folding argument, the prover sends  $a_i = f_{i,R}(\mathbf{x}_{i,L})$  and  $b_i = f_{i,L}(\mathbf{x}_{i,R})$ . As such, as discussed above,  $a_i$  and  $b_i$  are not recyclable. Note that there are cases when the verifier *already knows* the values of  $a_i$  and  $b_i$  that it should expect to receive based on the functions  $f_{i,L}, f_{i,R}$ ; for example, if either is the zero function. More generally, the verifier might be able to predict the value of  $a_i, b_i$  given  $f, a_{i-1}, b_{i-1}, c_{i-1}, a_{i-2}, b_{i-2}, c_{i-2} \dots$ . As such,  $a_i, b_i$  are not generally recyclable. However, since  $f$  is a linear form, we observe that the possible values of  $a_i, b_i$  correspond to the solutions of a linear system in the coefficients of  $f$  and the challenges so far. As such, they are either marginally uniform, or there is an efficient algorithm determining their unique assignment. Hence, we propose to modify the protocol to have the prover send *uniform* elements  $a_i$  or  $b_i$  when their “correct” value can already be determined by the verifier. The verifier can simply *ignore* these elements when the “correct” value is already determined. It is easy to see that this does not affect soundness or zero-knowledge of Compressed  $\Sigma$ -protocols.

We give a complete description of the protocol, including these modifications in the full version [28]. To capture our second modification, we define a function `UniqueOrRand` that is used to determine values  $a_i$  and  $b_i$  in each

folding. In particular, for each folding (to compute  $a_i, b_i$ ), it takes the following inputs: the function  $f$ , evaluation  $y = f(\mathbf{x})$ , previously computed values and challenges  $a_{i-1}, b_{i-1}, c_{i-1}, a_{i-2}, b_{i-2}, c_{i-2} \dots$  and  $f_{1,R}(\mathbf{x}_{iL})$  (when computing  $a_i$ ) or  $f_{1,L}(\mathbf{x}_{iR})$  (when computing  $b_i$ ). UniqueOrRand checks if the values  $a_i$  and  $b_i$  are already determined based on previous computed values and challenges — in which case it outputs a random value — else, it outputs  $f_{1,R}(\mathbf{x}_{iL})$  for  $a_i$  and  $f_{1,L}(\mathbf{x}_{iR})$  for  $b_i$ . We are now ready to describe how to speed-stack Compressed  $\Sigma$ -Protocols and prove the following theorem, setting  $M_{\text{RAND}}^{\text{COMP}} = (Q_1, A_1, B_1, a_1, b_1, \dots, A_{\mu-1}, B_{\mu-1}, a_{\mu-1}, b_{\mu-1}, \mathbf{z})$  for notational convenience.

**Theorem 8.** *Compressed  $\Sigma$ -protocols [4], denoted as  $\Pi_{\text{compressed}}$ , is stackable.*

We give a proof for Theorem 8 in the full version of the paper. Combining Theorems 8 and 2, we get the following Corollary.

**Corollary 1.** *Let  $\Pi_{\text{speed-compressed}}$  be output of the compiler in Fig. 2 recursively applied to  $\Pi_{\text{compressed}}$  using  $\mathcal{S}_{\text{RAND}}^{\text{COMP}}$  and  $\mathcal{S}_{\text{DET}}^{\text{COMP}}$  as defined in the proof of Theorem 8. Then  $\Pi_{\text{speed-compressed}}$  is a stackable ZK-IP for  $\mathcal{R}_{\text{dis-compressed}}$  with logarithmic communication complexity, and prover computational complexity  $O(\text{Time}(\Pi_{\text{compressed}}) + \ell \cdot \text{Time}(\mathcal{S}_{\text{DET}}^{\text{COMP}}))$ .*

**Efficiency of Speed-Stacked Compressed  $\Sigma$ -Protocols.** Our goal in stacking Compressed  $\Sigma$ -Protocols is to minimize the number of group operations that the prover must perform when proving a disjunctive statement, as group operations are typically significantly more expensive than field operations. Based on our compiler, it is easy to see that we get the most savings when the linear form  $f$  is actually a homomorphism from one field to another field. In that case the vast majority of the group operations are only necessary in the active clause. Concretely, the prover’s computational cost for running the compiled protocol is  $\text{Time}(\Pi_{\text{compressed}}) + \ell \cdot \text{Time}(\mathcal{S}_{\text{DET}}^{\text{COMP}}) + \text{Time}(\text{Gen}) + \text{Time}(\text{EquivCom}) + \text{Time}(\text{Equiv})$ . In this case, in  $\mathcal{S}_{\text{DET}}^{\text{COMP}}$ , the prover computes only 1 exponentiation and 1 group operation ( $T := Q_1 P^{-c}$ ). If we consider the commitment scheme proposed by Goel et al. [26], both key generation and committing require  $\ell$  exponentiations and group operations, while equivocation requires only field operations. Thus the overhead (when counting group operations) introduced from running a disjunction with  $\ell$  clauses is only  $2\ell$  exponentiation and  $2\ell$  group operations. Importantly all the multi-exponentiations resulting from folding  $\mathbf{g}$  and computing the  $A_i, B_i$  can be completely avoided.

We note that our modifications to the protocol do introduce some overheads. Namely, the verifier (and thus the deterministic simulator) need to decide when a message is already uniquely determined. This computation requires attempting to solve the system of equations for the particular value  $a_i, b_i$ . The verifier can simply do this using Gauss-Jordan elimination, which will take  $N \log^2(N)$  field operations.

**Extension to Circuit Satisfiability.** Due to space constraints, we discuss the details of speed-stacking the circuit satisfiability in the full paper [28].

**Acknowledgements.** The first author was supported in part by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. This work was done in part while the first author was a student at Johns Hopkins University and while they were visiting University of California, Berkeley. The second author is funded by Concordium Blockchain Research Center, Aarhus University, Denmark. The third author is supported by the National Science Foundation under Grant #2030859 to the Computing Research Association for the CIFellows Project and is supported by DARPA under Agreement No. HR00112020021. This work was completed in part while the fourth author was at Boston University and was supported by DARPA under Agreement No. HR00112020023. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

## References

1. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of- $n$  signatures from a variety of keys. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 415–432. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-36178-2\\_26](https://doi.org/10.1007/3-540-36178-2_26)
2. Agrawal, S., Raghuraman, S.: KV<sub>a</sub>C: key-value commitments for blockchains and beyond. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 839–869. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-64840-4\\_28](https://doi.org/10.1007/978-3-030-64840-4_28)
3. Attema, T., Cramer, R.: Compressed  $\Sigma$ -protocol theory and practical application to plug & play secure algorithmics. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 513–543. Springer, Cham, August 2020. [https://doi.org/10.1007/978-3-030-56877-1\\_18](https://doi.org/10.1007/978-3-030-56877-1_18)
4. Attema, T., Cramer, R., Fehr, S.: Compressing proofs of  $k$ -out-of- $n$  partial knowledge. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 65–91. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84259-8\\_3](https://doi.org/10.1007/978-3-030-84259-8_3)
5. Attema, T., Cramer, R., Kohl, L.: A compressed  $\Sigma$ -protocol theory for lattices. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 549–579. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84245-1\\_19](https://doi.org/10.1007/978-3-030-84245-1_19)
6. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in poly-logarithmic time. In: 23rd ACM STOC, pp. 21–31. ACM Press, May 1991. <https://doi.org/10.1145/103418.103428>
7. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac’ n’ Cheese: zero-knowledge proofs for Boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 92–122. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84259-8\\_4](https://doi.org/10.1007/978-3-030-84259-8_4)
8. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018). <https://eprint.iacr.org/2018/046>
9. Ben-Sasson, E., et al.: Zerocash: decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 459–474. IEEE Computer Society Press, May 2014. <https://doi.org/10.1109/SP.2014.36>
10. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17653-2\\_4](https://doi.org/10.1007/978-3-030-17653-2_4)

11. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53644-5\\_2](https://doi.org/10.1007/978-3-662-53644-5_2)
12. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: Fu, K., Jung, J. (eds.) USENIX Security 2014, pp. 781–796. USENIX Association, August 2014
13. Ben-Sasson, E., Goldberg, L., Kopparty, S., Saraf, S.: DEEP-FRI: sampling outside the box improves soundness. In: Vidick, T. (ed.) ITCS 2020, vol. 151, pp. 5:1–5:32. LIPIcs, January 2020. <https://doi.org/10.4230/LIPIcs.ITCS.2020.5>
14. Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 168–197. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-64378-2\\_7](https://doi.org/10.1007/978-3-030-64378-2_7)
15. Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Time- and space-efficient arguments from groups of unknown order. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 123–152. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84259-8\\_5](https://doi.org/10.1007/978-3-030-84259-8_5)
16. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 561–586. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-26948-7\\_20](https://doi.org/10.1007/978-3-030-26948-7_20)
17. 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, Part II. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49896-5\\_12](https://doi.org/10.1007/978-3-662-49896-5_12)
18. Borodin, A., Moenck, R.: Fast modular transforms. *J. Comput. Syst. Sci.* **8**(3), 366–386 (1974)
19. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press, May 2018. <https://doi.org/10.1109/SP.2018.00020>
20. Chiesa, A., Ojha, D., Spooner, N.: FRACTAL: post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 769–793. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45721-1\\_27](https://doi.org/10.1007/978-3-030-45721-1_27)
21. Ciampi, M., Persiano, G., Scafuro, A., Siniscalchi, L., Visconti, I.: Online/offline OR composition of sigma protocols. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 63–92. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49896-5\\_3](https://doi.org/10.1007/978-3-662-49896-5_3)
22. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994). [https://doi.org/10.1007/3-540-48658-5\\_19](https://doi.org/10.1007/3-540-48658-5_19)
23. swisspost evoting: E-voting system 2019. <https://gitlab.com/swisspost-evoting/e-voting-system-2019> (2019)
24. Garay, J.A., MacKenzie, P., Yang, K.: Strengthening zero-knowledge protocols using signatures. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 177–194. Springer, Heidelberg (2003). [https://doi.org/10.1007/3-540-39200-9\\_11](https://doi.org/10.1007/3-540-39200-9_11)
25. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EURO-

- CRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-38348-9\\_37](https://doi.org/10.1007/978-3-642-38348-9_37)
26. Goel, A., Green, M., Hall-Andersen, M., Kaptchuk, G.: Stacking Sigmas: a framework to compose  $\Sigma$ -protocols for disjunctions. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II, pp. 458–487. LNCS, Springer, Heidelberg, June 2022. [https://doi.org/10.1007/978-3-031-07085-3\\_16](https://doi.org/10.1007/978-3-031-07085-3_16)
  27. Goel, A., Hall-Andersen, M., Hegde, A., Jain, A.: Secure multiparty computation with free branching. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part I, pp. 397–426. LNCS, Springer, Heidelberg, June 2022. [https://doi.org/10.1007/978-3-031-06944-4\\_14](https://doi.org/10.1007/978-3-031-06944-4_14)
  28. Goel, A., Hall-Andersen, M., Kaptchuk, G., Spooner, N.: Speed-stacking: fast sub-linear zero-knowledge proofs for disjunctions. IACR Cryptol. ePrint Arch, p. 1419 (2022). <https://eprint.iacr.org/2022/1419>
  29. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: 27th FOCS, pp. 174–187. IEEE Computer Society Press, October 1986. <https://doi.org/10.1109/SFCS.1986.47>
  30. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: 17th ACM STOC, pp. 291–304. ACM Press, May 1985. <https://doi.org/10.1145/22145.22178>
  31. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49896-5\\_11](https://doi.org/10.1007/978-3-662-49896-5_11)
  32. Groth, J., Kohlweiss, M.: One-out-of-many proofs: or how to leak a secret and spend a coin. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 253–280. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46803-6\\_9](https://doi.org/10.1007/978-3-662-46803-6_9)
  33. Heath, D., Kolesnikov, V.: Stacked garbling. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 763–792. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-56880-1\\_27](https://doi.org/10.1007/978-3-030-56880-1_27)
  34. Heath, D., Kolesnikov, V.: Stacked garbling for disjunctive zero-knowledge proofs. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 569–598. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45727-3\\_19](https://doi.org/10.1007/978-3-030-45727-3_19)
  35. Heath, D., Kolesnikov, V.: LogStack: stacked garbling with  $O(b \log b)$  computation. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021, Part III. LNCS, vol. 12698, pp. 3–32. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77883-5\\_1](https://doi.org/10.1007/978-3-030-77883-5_1)
  36. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 955–966. ACM Press, November 2013. <https://doi.org/10.1145/2508859.2516662>
  37. 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>
  38. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC, pp. 723–732. ACM Press, May 1992. <https://doi.org/10.1145/129712.129782>

39. Kilian, J.: On the complexity of bounded-interaction and noninteractive zero-knowledge proofs. In: 35th FOCS, pp. 466–477. IEEE Computer Society Press, November 1994. <https://doi.org/10.1109/SFCS.1994.365744>
40. Kim, A., Liang, X., Pandey, O.: A new approach to efficient non-malleable zero-knowledge. In: Dodis, Y., Shrimpton, T. (eds.) *Advances in Cryptology - CRYPTO 2022–42nd Annual International Cryptology Conference, CRYPTO 2022*, Santa Barbara, CA, USA, 15–18 August 2022, Proceedings, Part IV. *Lecture Notes in Computer Science*, vol. 13510, pp. 389–418. Springer (2022). [https://doi.org/10.1007/978-3-031-15985-5\\_14](https://doi.org/10.1007/978-3-031-15985-5_14)
41. Kolesnikov, V.: Free IF: How to omit inactive branches and implement  $\mathcal{S}$ -universal garbled circuit (almost) for free. In: Peyrin, T., Galbraith, S. (eds.) *ASIACRYPT 2018*, Part III. *LNCS*, vol. 11274, pp. 34–58. Springer, Cham (2018). [https://doi.org/10.1007/978-3-030-03332-3\\_2](https://doi.org/10.1007/978-3-030-03332-3_2)
42. Lipmaa, H., Mohassel, P., Sadeghian, S.: Valiant’s universal circuit: Improvements, implementation, and applications. *Cryptology ePrint Archive*, Report 2016/017 (2016). <https://eprint.iacr.org/2016/017>
43. Liu, H., Yu, Yu., Zhao, S., Zhang, J., Liu, W., Hu, Z.: Pushing the limits of Valiant’s universal circuits: simpler, tighter and more compact. In: Malkin, T., Peikert, C. (eds.) *CRYPTO 2021*, Part II. *LNCS*, vol. 12826, pp. 365–394. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84245-1\\_13](https://doi.org/10.1007/978-3-030-84245-1_13)
44. Micali, S.: CS proofs (extended abstracts). In: 35th FOCS, pp. 436–453. IEEE Computer Society Press, November 1994. <https://doi.org/10.1109/SFCS.1994.365746>
45. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC, pp. 49–62. ACM Press, June 2016. <https://doi.org/10.1145/2897518.2897652>
46. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) *CRYPTO 1989*. *LNCS*, vol. 435, pp. 239–252. Springer, New York (1990). [https://doi.org/10.1007/0-387-34805-0\\_22](https://doi.org/10.1007/0-387-34805-0_22)
47. Valiant, L.G.: Universal circuits (preliminary report). In: *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, p. 196–203. STOC 1976, Association for Computing Machinery, New York (1976). <https://doi.org/10.1145/800113.803649>
48. Zaverucha, G.: The picnic signature algorithm. Technical report (2020). <https://raw.githubusercontent.com/microsoft/Picnic/master/spec/spec-v3.0.pdf>





# Proof-Carrying Data from Arithmetized Random Oracles

Megan Chen<sup>1</sup>, Alessandro Chiesa<sup>2</sup>, Tom Gur<sup>3</sup>, Jack O'Connor<sup>3(✉)</sup>,  
and Nicholas Spooner<sup>3</sup>

<sup>1</sup> Boston University, Boston, USA

<sup>2</sup> EPFL, Lausanne, Switzerland

<sup>3</sup> University of Warwick, Coventry, UK

Jack.O-Connor@warwick.ac.uk

**Abstract.** Proof-carrying data (PCD) is a powerful cryptographic primitive that allows mutually distrustful parties to perform distributed computation in an efficiently verifiable manner. Known constructions of PCD are obtained by recursively-composing SNARKs or related primitives. SNARKs with desirable properties such as transparent setup are constructed in the random oracle model. However, using such SNARKs to construct PCD requires heuristically instantiating the oracle and using it in a non-black-box way. [CCS22] constructed SNARKs in the low-degree random oracle model, circumventing this issue, but instantiating their model in the real world appears difficult.

In this paper, we introduce a new model: the *arithmetized random oracle model* (AROM). We provide a plausible standard-model (software-only) instantiation of the AROM, and we construct PCD in the AROM, given only a standard-model collision-resistant hash function. Furthermore, our PCD construction is for arbitrary-depth compliance predicates. We obtain our PCD construction by showing how to construct SNARKs in the AROM for computations that query the oracle, given an accumulation scheme for oracle queries in the AROM. We then construct such an accumulation scheme for the AROM.

We give an efficient “lazy sampling” algorithm (an *emulator*) for the ARO up to some error. Our emulator enables us to prove the security of cryptographic constructs in the AROM and that zkSNARKs in the ROM also satisfy zero-knowledge in the AROM. The algorithm is non-trivial, and relies on results in algebraic query complexity and the combinatorial nullstellensatz.

**Keywords:** proof-carrying data · random oracle model · arithmetization

## 1 Introduction

Proof-carrying data (PCD) [CT10] is a powerful cryptographic primitive that allows mutually distrustful parties to perform distributed computation in an efficiently verifiable manner. The notion of PCD generalizes incrementally-verifiable computation (IVC) [Val08] and has recently found exciting applications in enforcing language semantics [CTV13], verifiable MapReduce computations [CTV15], image authentication [NT16], verifiable registries [TFZ+22], blockchains [Mina, KB20, BMRS20, CCDW20], and more.

© International Association for Cryptologic Research 2023

C. Hazay and M. Stam (Eds.): EUROCRYPT 2023, LNCS 14005, pp. 379–404, 2023.

[https://doi.org/10.1007/978-3-031-30617-4\\_13](https://doi.org/10.1007/978-3-031-30617-4_13)

All known PCD constructions (and practical IVC constructions) are obtained via *recursive proof composition*, a general framework for building PCD from simpler primitives such as SNARKs [BCCT13, BCTV14, COS20] or accumulation schemes [BGH19, BCMS20, BDFG21, BCL+21, KST22]. While the specific constructions differ, the high-level idea remains the same: to prove the correctness of  $t$  steps of computation given proof of correctness for  $t - 1$  steps, one proves that “the  $t$ -th step is correct *and* there exists a valid proof for the first  $t - 1$  steps”.

The statement that “there exists a valid proof” refers to the *verifier* of the underlying SNARK or accumulation scheme. As such, the resulting PCD scheme makes non-black-box use of the verifier for the underlying scheme. This leads to a significant theoretical problem when trying to prove security for constructions based on recursive composition: almost all known constructions of SNARKs, and all known constructions of accumulation schemes, are proven secure in the random oracle model (ROM). The random oracle is an inherently black-box object; in particular, it is believed that there is no “nontrivial” proof system for statements about the random oracle.

Most prior work in the area [COS20, BCMS20, BCL+21] avoids this problem using a *heuristic step*: they assume that there exists some concrete hash function such that replacing the random oracle with the hash function yields a secure SNARK or accumulation scheme in the standard model (without oracles), and then apply recursive composition to this heuristic scheme.

Two prior works [CT10, CCS22] propose a different approach: endow the random oracle with some additional structure. The PCD construction in [CT10] is in a model where the random oracle additionally *signs* its responses using a standard-model signature scheme; the verifier can then check query-answer pairs by verifying the signature rather than querying the oracle. Trading cryptographic structure for algebraic structure, [CCS22] construct PCD in the *low-degree random oracle model* (LDROM), where parties have access to a random low-degree multivariate polynomial.

Both of these oracle models can be instantiated using hardware tokens. Unfortunately, we do not have any standard model (i.e., software-only) instantiation of these oracles, even heuristically. This is in contrast to the (usual) random oracle model, where empirical evidence suggests that “natural” schemes remain secure provided the oracle is replaced with a suitably “random-looking” hash function [BR93]. Our goal in this work is to design a new oracle model that simultaneously achieves both desiderata: (a) there exists a PCD scheme in this model under standard assumptions; and (b) the oracle can be heuristically instantiated.

## 1.1 Our Results

In this work we introduce and study a new oracle model, the arithmetized random oracle model (AROM), which provides a random oracle and a corresponding “arithmetization” oracle. As in the standard ROM, the random oracle is an idealized model of some concrete hash function  $H$ . The arithmetization oracle is an idealized model of a certain *arithmetization* of  $H$ , which is a low-degree polynomial  $P_H$  that can be efficiently computed from the circuit of  $H$ . As such, the AROM has a plausible heuristic instantiation: replace the random oracle by  $H$  and the arithmetization oracle by  $P_H$ , for a suitable hash function  $H$ .

Our main result is a construction of PCD in the AROM, based on the [CCS22] construction of PCD in the LDROM. By instantiating the AROM with a suitable hash function, we obtain a candidate “real-world” construction of PCD. Formally, we prove the following theorem.

**Theorem 1** (informal). *There exists transparent<sup>1</sup> (zero-knowledge) PCD in the AROM (for computations in the AROM), assuming the existence of collision-resistant hash functions in the standard model.*

Our PCD construction is provably secure (in the AROM) for *all* efficient compliance predicates. This stands in contrast to all other constructions of PCD (with the exception of [CT10], but including [CCS22]), whose security proofs are limited to constant-depth recursion. This is because, like [CT10], our PCD construction preserves the straightline extraction property of the underlying SNARK.<sup>2</sup>

To prove our main theorem, we develop various tools for analyzing cryptographic constructions in the AROM. Our key result here is to show that the additional power provided by the AROM does not help the adversary win any game defined with respect to the random oracle alone.

**Theorem 2** (informal). *Any construction that is secure in the ROM is secure in the AROM.*

An immediate consequence of this theorem is that any construction that is secure in the standard model is secure in the AROM. In contrast, we do not know whether an analogous statement holds in the LDROM. We remark that this result is meaningful even outside the present context: it provides evidence that security in the ROM implies security against a specific type of non-black-box attack, namely, attacks that treat the *arithmetization* of the hash function as a black box.

**Comparison to Other Oracle Models.** As discussed above, both the ROM and the LDROM fall short of our goal. While the ROM has a well-established heuristic instantiation, it is unlikely to support a PCD scheme. PCD exists in the LDROM, but we do not know how to instantiate the oracle. The AROM offers, in some sense, the “best of both worlds”: a provable construction of PCD *and* a plausible heuristic instantiation. Moreover, the proposed instantiation of the AROM does not rely on any cryptography beyond “random-oracle-like” hash functions. As such, there are no barriers to implementing our scheme.

**Post-quantum Security.** Our scheme does not rely on any pre-quantum assumption; it is plausibly post-quantum secure. Moreover, it is conceivable that the scheme is in fact *provably* post-quantum secure in the “quantum-accessible” AROM; we leave this intriguing question to future work.

## 1.2 Related Work

**PCD and IVC in the ROM.** There is theoretical evidence that, unlike for SNARKs, there is no construction of PCD and IVC in the ROM (even allowing for additional

<sup>1</sup> The only setup required is a uniform reference string.

<sup>2</sup> Some other prior PCD constructions are also based on SNARKs with straightline extraction (e.g., [Val08, COS20]). However, this property is lost after the heuristic step is applied.

“mild” cryptographic assumptions like standard-model CRHs). First, [CL20] shows that the PCP theorem does not hold for various cryptographically relevant oracle models, such as the ROM and the LDROM. This suggests that succinct proofs for computations relative to these oracles may be out of reach. Nevertheless, [CCS22] shows that this is not the whole story by constructing SNARKs for LDROM computations, particularly PCD, from a cryptographic assumption. Second, [HN22] shows various impossibilities for IVC in the ROM. For example, if a particular type of commitment scheme exists, then zero-knowledge IVC (without a CRS) does not exist in the ROM. This result holds even if the IVC construction were to rely on “standard” cryptographic assumptions.<sup>3</sup>

**Pseudorandom Oracles.** [JLLW22] introduce the *pseudorandom oracle model* (PROM) and apply it towards obfuscation. Similarly to the AROM, the PROM aims to capture cryptographic schemes that make a non-black-box use of the random oracle. We outline the PROM and explain how it differs from the AROM.

The PROM is specified relative to a (standard model) pseudorandom function family  $F_k$ , and has two interfaces. The first accepts a key  $k$  and outputs a random handle  $h$  (and stores  $(h, k)$ ). The second accepts a handle  $h$  and an input  $x$  and outputs  $F_k(x)$ , where  $k$  is the key corresponding to  $h$ . By the security of the PRF, a party holding only  $h$  cannot distinguish the latter interface from a random oracle. On the other hand, a party holding the key  $k$  can use the circuit for  $F_k$  in a non-black-box way. [JLLW22] constructs ideal obfuscation from functional encryption in the PROM.

The key difference between the AROM and the PROM is that the PROM “separates” non-black-box and black-box access to the oracle. Specifically, non-black-box access to the PROM is available only to parties that know  $k$ , whereas random oracle security holds only against parties that do not know  $k$ . *In the AROM, there is no such asymmetry: all parties have the same access to the oracle.* This is important in the context of recursive composition (which we study) since completeness requires that both the prover and the verifier have non-black-box access to the oracle. Still, soundness relies on the security of the random oracle against the prover. It is an exciting open question to understand whether, despite this apparent barrier, recursive composition is possible in the PROM.

**Augmented Random Oracles.** [Zha22] defines the *augmented* random oracle model to analyze the resilience of cryptographic transformations in the ROM against uninstantiability results. While ideas about modeling non-black-box access to the random oracle (and the abbreviation “AROM”) are common to both the augmented ROM and the arithmetized ROM, the models are very different both technically and in their applications. We briefly summarize [Zha22] and then explain how our model differs.

Let  $ro$  denote the random oracle, and  $\Pi$  denote some protocol. A cryptographic transformation  $T$  usually comes with a guarantee like “if  $\Pi$  is a secure X, then  $T^{ro}(\Pi)$  is a secure Y”. An uninstantiability result for  $T$  typically shows that there exists some  $\Pi$  such that  $T^H(\Pi)$  is insecure for every polynomial-size circuit  $H$ . Known uninstantiability results use some non-black-box technique to provide a “trapdoor” that can be used with respect to any  $H$  but is useless for  $ro$ . The augmented ROM captures this

<sup>3</sup> The paper claims that this result holds for constructions that use *falsifiable* assumptions but does not show this explicitly. Nonetheless, one can check that the proof does work for “benign” cryptographic assumptions.

paradigm by requiring  $T^{\text{ro}}(H)$  to be secure even if  $H$  has access to an oracle  $M$  that provides some functionality permitted by non-black-box access to  $H$ , but with respect to  $\text{ro}$ . [Zha22] shows that key uninstanciability results for transformations (e.g., Fiat–Shamir for arguments [GK03]) lead to insecure protocols in the augmented ROM.

The augmented ROM is a tool for proving a stronger form of security for random oracle transformations. In particular, no “honest” scheme ever accesses the oracle  $M$ ; indeed, the oracle  $M$  is chosen adversarially (and may be trivial). On the other hand, in the arithmetized ROM, honest parties use the non-black-box access provided by the arithmetization oracle, whose functionality is (mostly) fixed by the model itself.

## 2 Techniques

Recall that our goal in this work is to construct proof-carrying data (PCD). Our approach follows the widely-used template of *recursive proof composition*. However, our setting imposes several technical and conceptual challenges. We begin by outlining a vital issue in proving security for this type of construction, which our work seeks to address.

Recursive proof composition refers to a set of techniques that enable the construction of PCD (and IVC) from SNARKs or accumulation schemes. With few notable exceptions (e.g., [Gro16]), all constructions of SNARKs and accumulation schemes rely on the Fiat–Shamir heuristic, which converts an interactive public-coin argument system into a non-interactive argument via a cryptographic hash function  $H$ . For all of these SNARK constructions, it is unknown whether this heuristic can be realized from any concrete (i.e., falsifiable) cryptographic assumption; indeed, there is evidence that this may not be possible [GW11]. However, we can prove these schemes secure in the ROM, treating the hash function  $H$  as a truly random function  $\text{ro}$  to which the adversary has black-box access.

This leads to a fundamental tension in proving security for the recursive composition of these protocols. On the one hand, to prove security for the protocol itself, we assume that the adversary treats the hash function  $H$  as a black box. On the other hand, when recursively composing, the *honest* protocol treats  $H$  in a non-black-box way: specifically, as a concrete polynomial-size circuit. The prior work [CL20, HN22] discussed in Sect. 1.2 suggests that non-black-box use of  $H$  may be necessary to achieve PCD (and IVC).

### 2.1 Starting Point: The Low-Degree Random Oracle Model

The work of [CCS22] addresses the aforementioned tension by introducing a new oracle model called the *low-degree random oracle model* (LDROM). They then show how to construct PCD via recursive composition in the LDROM (i.e., using the oracle as a black box).

In the LDROM, all parties have oracle access to a uniformly random low-degree multivariate polynomial  $\hat{\rho}: \mathbb{F}^m \rightarrow \mathbb{F}$ . Restricting  $\hat{\rho}$  to  $\{0, 1\}^m \subseteq \mathbb{F}^m$  recovers the usual random oracle, and [CCS22] show that relevant security properties of the random oracle continue to hold in the LDROM; in particular, Micali’s SNARK [Mic00] is secure in the LDROM. Unlike the random oracle, the LDROM admits a *query accumulation scheme*: a verifier, with the help of an untrusted accumulation proof, can check the correctness

of  $n$  queries to  $\hat{\rho}$  using only  $O(1)$  queries to  $\hat{\rho}$ . [CCS22] construct such an accumulation scheme and use it to build PCD.

**Instantiating the LDROM.** [CCS22] observe that the LDROM can be instantiated using a hardware token that implements the structured PRF of [BGV11]. Of course, schemes involving hardware tokens have significant drawbacks; finding a plausible “software-only” instantiation would be much preferable. [CCS22] suggest a natural strategy: given a “random-oracle-like” hash function  $H$ , convert it into an arithmetic circuit gate-by-gate. Such a circuit does define a polynomial with which we could instantiate the LDROM. Unfortunately, as noted in [CCS22], for widely-used hash functions, the degree of this polynomial will be large (at least  $2^{25}$ ). Since the complexity of the verifier in the query accumulation scheme is linear in the degree of the oracle, the resulting PCD scheme would be prohibitively expensive.

## 2.2 The Arithmetized Random Oracle Model

Given the above difficulty, a natural next step is to consider techniques for *reducing the degree* of the resulting arithmetic circuit. Since the degree of an arithmetic circuit grows exponentially in its depth, a natural approach is to try to reduce the depth of the circuit for  $H$ . This can be achieved via the well-known NP reduction from circuit satisfiability to 3-SAT (a depth-two formula). The output of the reduction is a boolean formula  $\Phi_H$  with the following property: there is an efficiently computable *witness function*  $W_H$  such that

$$\Phi_H(x, y, z) = \begin{cases} 1 & \text{if } H(x) = y \text{ and } W_H(x) = z \\ 0 & \text{otherwise} \end{cases}.$$

Converting  $\Phi_H$  into an arithmetic formula (gate-by-gate) yields a polynomial  $P_H$  of total degree  $O(|H|)$  that agrees with  $\Phi_H$  on boolean inputs.

$P_H$  is *not* a low-degree extension of  $H$  (rather of  $\Phi_H$ ) and so this is not a candidate instantiation of the LDROM. As we note later, however, the low-degree structure of  $P_H$  will nonetheless allow us to build a query accumulation scheme, inspired by that of [CCS22]. Moreover, the statement “ $H(x) = y$ ” can be verified by querying  $P_H$  only, given  $z$  as a witness. It is therefore plausible that, following the template developed in the prior work, we can obtain a secure construction of PCD that makes only black-box use of  $H$  and  $P_H$ .

Of course, given the current state of knowledge, we can only hope to prove that this PCD scheme is secure in some idealized model. In particular, we would like to model  $H$  as a random oracle. It is then necessary to answer the question: *if  $H$  is a random oracle, what should  $P_H$  look like?* A central modeling contribution of our work is to propose an answer to this question.

**A New Oracle Model: The AROM.** We refer to our proposed oracle model as the *arithmetized random oracle model* (AROM). Before presenting the model, we discuss two key modeling challenges that arise. Both relate to the fact that the black-box behavior of  $P_H$  depends in a non-black-box way on  $H$ .

- **Challenge #1:  $W_H$  is circuit-dependent.** For a concrete circuit  $H$  and input  $x$ ,  $W_H(x)$  is a vector representing the assignment to the internal wires of  $H$  on input

$x$ . This of course depends on the size and structure of the circuit for  $H$ , which is no longer meaningful when  $H$  is replaced by a random oracle. We handle this conservatively, by allowing  $W_H$  to be adversarial. That is, we require that completeness, soundness, and zero-knowledge hold *regardless* of the choice of  $W_H$ , which we allow to depend on  $x$  and the random oracle, and may even itself be randomized.

There is, however, an important caveat. While we allow our  $W_H$  to depend on the random oracle, we must restrict this dependency; otherwise, the adversary could use  $W_H$  to learn information that it cannot otherwise obtain (e.g.,  $W_H$  could encode a collision in  $H$ ). Similarly, if  $W_H$  is computationally unbounded, the adversary could use it to break standard-model cryptography. As such, we restrict  $W_H$  to have an efficient implementation (in particular, it can only make polynomially-many queries to  $H$ ).

- **Challenge #2:  $P_H$  is not the unique extension.** Even after we have fixed  $W_H$  (and hence  $\Phi_H$ ),  $P_H$  has a huge number of remaining degrees of freedom. This is because it is of individual degree larger than 1, but its behavior is specified only on boolean inputs. This is a more challenging issue to resolve: letting  $P_H$  be chosen adversarially from the set of extensions of  $\Phi_H$  would make the adversary unrealistically powerful (see Remark 1). Instead, we model  $P_H$  as a uniformly random polynomial of the appropriate degree whose restriction to the hypercube is  $\Phi_H$ . We propose that this captures the inability of the adversary to leverage the structure of  $H$  (and hence  $P_H$ ) in breaking security. We leave to future work the question of whether this modeling choice can be weakened (again see Remark 1).

We now give an informal definition of the AROM; for details see Sect. 4. In the AROM, all parties (honest and malicious) have access to three oracles ( $\text{ro}$ ,  $\text{wo}$ ,  $\widehat{\text{vo}}$ ):

- a *random oracle*  $\text{ro}: \{0, 1\}^m \rightarrow \{0, 1\}^\lambda$  drawn uniformly at random;
- a *witness oracle*  $\text{wo}: \{0, 1\}^m \rightarrow \{0, 1\}^w$  that is an arbitrary PPT-computable function (see below);
- an *extended verification oracle* (arithmetization oracle)  $\widehat{\text{vo}}: \mathbb{F}^{m+\lambda+w} \rightarrow \mathbb{F}$  that is a random extension of individual degree  $d \geq 2$  of the verification oracle  $\text{vo}: \{0, 1\}^{m+\lambda+w} \rightarrow \{0, 1\}$  defined as follows:

$$\text{vo}(x, y, z) := \begin{cases} 1 & \text{if } \text{ro}(x) = y \text{ and } \text{wo}(x) = z \\ 0 & \text{otherwise} \end{cases} .$$

We discuss each oracle in turn.

- The random oracle  $\text{ro}$  models the hash function  $H$ , as in the standard ROM.
- The witness oracle  $\text{wo}$  models the witness function  $W_H$ . It is defined via a polynomial-size oracle circuit  $B$  chosen arbitrarily before the oracle is sampled. On a query  $x$ ,  $\text{wo}$  outputs  $B^{\text{ro}}(x, \mu_x)$  where  $\mu_x$  is sampled uniformly at random (and is not resampled if  $x$  is queried again). The inclusion of  $\mu_x$  allows our definition to subsume, e.g., modeling  $W_H$  as a random oracle. The efficiency requirement is necessary to allow for efficient simulation of  $\text{wo}$  (it prevents  $\text{wo}$  from being used to break standard-model cryptography).

- The verification function  $\text{vo}$  models the boolean formula  $\Phi_H$ . Indeed, the definition of  $\text{vo}$  is directly obtained from the definition of  $\Phi_H$  by replacing  $H$  with  $\text{ro}$  and  $W_H$  with  $\text{wo}$ .
- The extended verification oracle  $\widehat{\text{vo}}$  models the polynomial  $P_H$ . The requirement that  $d \geq 2$  arises from a technical concern: as noted in [JKRS09], access to the unique multilinear ( $d = 1$ ) extension of a function can be surprisingly powerful. (E.g., an adversary with access to the multilinear extension of  $\widehat{\text{vo}}$  can efficiently invert  $\text{ro}$ , see Remark 1.) Requiring  $d \geq 2$  avoids this issue and is sufficient for our security proofs. In any case, we want to match the degree of  $\widehat{\text{vo}}$  to that of  $P_H$  for some concrete hash function  $H$ , and the degree of  $P_H$  will be at least 2 in each variable.<sup>4</sup>

A construction that makes black-box use of  $H, W_H, \Phi_H$  can be analyzed in the AROM as suggested by the above discussion: replace  $H$  with  $\text{ro}$ ,  $W_H$  with  $\text{wo}$ , and  $P_H$  with  $\widehat{\text{vo}}$  (with matching degree bound  $d$ ).

In Sect. 2.3 we describe our construction of PCD in the AROM. This construction relies on a “lazy sampling” procedure for the AROM, a key technical contribution that we describe in Sect. 2.4.

**AROM vs. LDROM.** Superficially the AROM and LDROM seem quite similar; indeed, they both aim to capture some arithmetization of the random oracle. However, there are notable differences between the two models, even putting aside the differing instantiability considerations. We highlight a few such differences.

- The LDRO is a low-degree extension over a field  $\mathbb{F}$  of a random function  $\{0, 1\}^m \rightarrow \mathbb{F}$ . Hence the security of the LDRO as a random oracle depends on  $|\mathbb{F}|$ . The ARO decouples the choice of  $\mathbb{F}$  from the random oracle: one may choose the codomain  $\{0, 1\}^\lambda$  of  $\text{ro}$  independently from the field  $\mathbb{F}$  over which  $\widehat{\text{vo}}$  is defined. The security of  $\text{ro}$  (even in the presence of  $\widehat{\text{vo}}$ ) depends only on  $\lambda$ . That said, in *both* the LDROM and the AROM, the security of their respective query accumulation schemes depends on  $|\mathbb{F}|$ .
- The LDRO is a linear code random oracle; i.e., it is sampled at random from a linear space over  $\mathbb{F}$ . The ARO is also sampled uniformly from some set, but this set does not form a linear space. This means that tools developed in [CCS22] for analyzing linear code random oracles do not directly apply. That said, the ARO does have *some* linear structure: the oracle  $\widehat{\text{vo}}$  is sampled uniformly from the (affine) space of low-degree extensions of  $\text{vo}$ . This fact will be useful for emulating the AROM.
- The LDRO has security properties (e.g. collision resistance, unconditional SNARKs) even when  $d = 1$  (i.e., it is a random *multilinear* polynomial). The ARO is not even one-way when  $d = 1$ .

*Remark 1 (choice of extension).* We set  $\widehat{\text{vo}}$  to be a random extension of  $\text{vo}$  of individual degree  $d \geq 2$ . We explain why setting  $\widehat{\text{vo}}$  to be an arbitrary extension of  $\text{vo}$  would grant the adversary too much power.

First consider the case when  $\widehat{\text{vo}}$  is the unique multilinear extension of  $\text{vo}$  ( $d = 1$ ). Given oracle access to a multilinear polynomial  $P$  over a field  $\mathbb{F}$  of characteristic different from 2, a single query to  $P$  suffices to efficiently evaluate the sum  $\sum_{x \in \{0,1\}^n} P(x)$

<sup>4</sup> The degree of a variable in  $P_H$  is equal to the number of clauses in  $\Phi_H$  in which it appears. Every wire appears in at least two clauses in  $\Phi_H$ : once as an output and once as an input.



[JKRS09]. We can use this capability and the structure of  $\text{vo}$  to invert  $\text{ro}$ : given a target image  $y \in \{0, 1\}^\lambda$ , perform a binary search for a preimage of  $y$  by evaluating the sum  $\sum_{x_1, z} \widehat{\text{vo}}((x_0, x_1), y, z)$  for different prefixes  $x_0$ .

Next consider the higher-degree case:  $\widehat{\text{vo}}$  is an *adversarially-chosen* extension of  $\text{vo}$  of degree  $d \geq 2$ . Given oracle access to a polynomial  $P$  of individual degree  $d$ , a single query to  $P$  suffices to efficiently evaluate the sum  $\sum_{x \in H^n} P(x)$  where  $H$  is a multiplicative subgroup of  $\mathbb{F}$  with  $|H| > d$  [CFS17, Lemma A.4]. Assume that  $\mathbb{F}$  has such a subgroup  $H$  of size  $d + 1$ . We can embed  $\{0, 1\}$  into  $H$  via an affine shift, and so we may abuse notation to assume  $\{0, 1\} \subseteq H$ . Choose  $\widehat{\text{vo}}$  to be the following extension of  $\text{vo}$ :  $\{0, 1\}^n \rightarrow \mathbb{F}$ :  $\widehat{\text{vo}}(x, y, z) = \text{vo}(x, y, z)$  for  $(x, y, z) \in \{0, 1\}^n$ , and  $\widehat{\text{vo}}(w) = 0$  for  $w \in H^n \setminus \{0, 1\}^n$ . Note that  $\widehat{\text{vo}}$  has individual degree  $|H| - 1 = d$ . Given this extension we can then use binary search as in the multilinear case to invert  $\text{ro}$ .

The above gives some justification for modelling  $\widehat{\text{vo}}$  as a *random* low-degree extension of  $\text{vo}$ . Of course, there are many choices that lie in between adversarial and random. For example, one could set  $\widehat{\text{vo}}$  to be drawn from an adversarially-chosen distribution with “enough” entropy. It is not clear, however, whether such a choice would be substantially closer to “reality” than our choice.

### 2.3 Building PCD Secure in the AROM

Prior work [CCS22] shows that to obtain PCD in an oracle model  $\mathcal{O}$ , it suffices to construct: (i) a SNARK for NP relative to  $\mathcal{O}$ ; and (ii) an accumulation scheme for  $\mathcal{O}$ -queries relative to  $\mathcal{O}$ . Further, the resulting PCD scheme is zero-knowledge if the SNARK and accumulation scheme also satisfy zero-knowledge. The PCD construction in the LDROM in [CCS22] follows by establishing these results for the LDROM. Similarly, our construction of PCD will follow by establishing these results for the AROM.

**(i) SNARKs in the AROM.** [CCS22] prove that Micali’s SNARK remains (information-theoretically) secure in the LDROM, via a rewinding argument. In the AROM, we show a much more general theorem.

**Theorem 3 (informal).** *Let  $p$  be a predicate that queries  $\text{ro}$ , and let  $\mathcal{A}$  be an algorithm querying  $(\text{ro}, \text{wo}, \widehat{\text{vo}})$  that outputs  $x$  satisfying  $p^\circ$  with probability  $\varepsilon$ . Then there is an algorithm  $\mathcal{B}$ , of similar efficiency to  $\mathcal{A}$ , that queries  $\text{ro}$  only and outputs  $x$  satisfying  $p^\circ$  with probability  $\varepsilon - \text{negl}(\lambda)$ .*

Theorem 3 follows directly from our emulator for  $\widehat{\text{vo}}$ , which we discuss further in Sect. 2.4. It is *not* known whether a similar result holds for the LDROM.

As an illustrative example, we can use Theorem 3 to prove that the ARO is collision-resistant. By applying Theorem 3 to the predicate  $p^\circ$  that, given  $(x, x') \in \{0, 1\}^m \times \{0, 1\}^m$ , checks that  $x \neq x'$  and  $\text{ro}(x) = \text{ro}(x')$ , we deduce that the ARO is collision-resistant from the fact that the RO is collision-resistant.

We use Theorem 3 to prove knowledge soundness and zero knowledge of Micali’s SNARK in the AROM.

- **Knowledge soundness.** We use Theorem 3 to prove that Micali’s SNARK is secure in the AROM, via a *straightline* extractor. Informally, since we can cast knowledge soundness of Micali’s SNARK as an oracle predicate  $p$ , any adversary  $\mathcal{A}$  that breaks

that security property in the AROM can be transformed via Theorem 3 into an adversary  $\mathcal{B}$  that breaks it in the ROM. We can then apply the straightline extractor for Micali’s SNARK to  $\mathcal{B}$ . Since  $\mathcal{B}$  invokes  $\mathcal{A}$  in a straightline manner, the resulting AROM extractor is also straightline.

- **Zero-knowledge.** We prove that Micali’s SNARK is zero knowledge in the AROM. Our zero knowledge simulator that *programs* the oracle; this is a commonality with the zero knowledge simulators for Micali’s SNARK in both the ROM and in the LDROM (see [CCS22]). To program the oracle, the simulator relies on a slightly stronger version of our emulator, which emulates oracle queries conditioned on an input list of (real) oracle query-answer pairs. Our hybrid argument invokes Theorem 3 and the Micali SNARK’s zero knowledge property in the ROM. Informally, we move between hybrids in the ROM vs AROM using Theorem 3, setting the predicate  $p$  to be any distinguisher between hybrids.

**(ii) An accumulation scheme for ARO queries.** The accumulation scheme for LDRO queries in [CCS22] is obtained by applying the Fiat–Shamir transformation to the (interactive public-coin) query reduction protocol of [KR08]. We follow the same template in the case of the ARO. The first observation is that it suffices to accumulate queries to  $\widehat{vo}$  only, because a query to  $ro$  or  $wo$  can be verified via a query to  $\widehat{vo}$ .<sup>5</sup>

The [KR08] query reduction protocol itself works for any low-degree polynomial: in particular, for  $\widehat{vo}$ . As in [CCS22], the central challenge is showing soundness of the Fiat–Shamir transformation in this setting. Note that here we *cannot* appeal to our general theorem above because the verification predicate queries  $\widehat{vo}$ .

The soundness of our accumulation scheme is captured by a *zero-finding game* (ZFG). First explicitly described by [BCMS20], the most basic form of a ZFG challenges the adversary to output a commitment  $cm$  (under a standard-model commitment scheme) to a low-degree polynomial  $f \not\equiv 0$  such that  $f(ro(cm)) = 0$ . Intuitively this is hard because  $f$  is fixed by  $cm$  before  $ro(cm)$  is known, and so the probability that  $f(ro(cm)) = 0$  cannot be much larger than the probability that  $f(\alpha) = 0$  for a random  $\alpha \in \mathbb{F}$ , which is negligible for large fields. [CCS22] shows that a more general version of the ZFG holds in the LDROM, where the ZFG polynomial may depend in a restricted way on the LDRO itself. That is, they show that it is hard to find a commitment  $cm$  to polynomials  $f, g$  such that  $f - \hat{p} \circ g \not\equiv 0$  but  $(f - \hat{p} \circ g)(\hat{p}(cm)) = 0$ .

The security of our construction depends on the hardness of a similar problem in the AROM, captured by the following lemma.

**Lemma 1** (informal). *It is hard for any polynomial-size adversary with access to the ARO ( $ro, wo, \widehat{vo}$ ) to find a commitment  $cm$  to a pair of low-degree polynomials  $f, g$  such that  $f - \widehat{vo} \circ g \not\equiv 0$  but  $(f - \widehat{vo} \circ g)(ro(cm)) = 0$ .*

We prove Lemma 1 by adapting the proof of the ZFG in [CCS22]. The proof relies on a *forking lemma* in the LDROM, which in turn relies on the ability to efficiently simulate the oracle in order to sample a forking transcript. For the AROM, we will rely on the emulator described in Sect. 2.4. The proof proceeds as follows. Looking ahead,

<sup>5</sup> Recall that  $ro(x) = y$  and  $wo(x) = z$  if and only if  $\widehat{vo}(x, y, z) = 1$ .

we note that the emulator maintains a polynomial  $P$  that it uses to answer queries to  $\widehat{v}_0$ . We show that the adversary cannot win the ZFG when  $\widehat{v}_0$  is replaced by  $P$ . This argument uses the forking lemma with respect to the emulator, and follows [CCS22], with one difference: this approach doesn't require a bespoke forking lemma as in [CCS22], and can be carried out using a general forking lemma [BN06, Lemma 1]. This general forking lemma is designed for random oracle adversaries, however, as we have already replaced  $\widehat{v}_0$  with the emulator we can “perfectly emulate”  $P$  using the emulator. Further, we can perfectly emulate  $w_0$  using the witness circuit  $B$ . This allows us to reduce the ZFG adversary to a random oracle adversary and thus apply the general forking lemma. Then, since the emulator is statistically indistinguishable from  $\widehat{v}_0$ , the adversary cannot win the original ZFG.

Before we describe our emulator, we discuss an important feature of our PCD construction.

**Extraction and PCD Depth.** Almost all constructions of PCD suffer from the “extractor blowup” problem. To obtain a PCD transcript of depth  $d$ , we apply the SNARK extractor to itself  $d$  times. If the extractor corresponding to a size- $S$  adversary is of size  $S^c$ , then the final extractor size is  $s^{c^d}$ , where  $s$  is the size of the original PCD adversary. As a result, one obtains meaningful security guarantees when  $d$  is a constant.

There is a single construction that does not suffer from this issue: the construction of [CT10]. This is because their SNARK (in their signed random oracle model) is straightline (or “list”) extractable. Micali’s SNARK is also straightline extractable in the ROM [Val08]. Of course, after heuristically instantiating the oracle there is no longer any notion of “straightline”. On the other hand, we can easily show that Micali’s SNARK is straightline extractable in the AROM. (We do not know how to show this in the LDROM; [CCS22] instead gives a rewinding extractor for Micali’s SNARK.)

As a result, our PCD construction is *secure for arbitrary recursion depth*.

## 2.4 Emulation of the ARO

As discussed in Sect. 2.3, we aim to construct PCD in the AROM by proving that cryptographic properties in the ROM, specifically knowledge soundness and zero-knowledge of the Micali SNARK, also hold in the AROM. To this end, we design an efficient algorithm  $\mathcal{M}$  that answers queries in a way that is statistically indistinguishable from answers of the ARO. We refer to such an algorithm as an *emulator*  $\mathcal{M}$  for the AROM.<sup>6</sup>

Recall that the ARO consists of a tuple of oracles  $(ro, wo, \widehat{v}_0)$ . Our emulator  $\mathcal{M}$  achieves a special (stronger) type of emulation: given oracle access to some  $ro$  and  $wo$ ,  $\mathcal{M}$  can efficiently emulate  $\widehat{v}_0$  drawn from the ARO distribution *conditioned* on  $(ro, wo)$ . We use this type of emulation to prove Theorem 3.

**Lemma 2 (informal).** *There exists a probabilistic algorithm  $\mathcal{M}$  such that for every security parameter  $\lambda \in \mathbb{N}$ , query bound  $t \in \mathbb{N}$ , and  $t$ -query adversary,*

$$\left| \Pr_{(ro, wo, \widehat{v}_0) \leftarrow \mathcal{O}(\lambda)} \left[ \mathcal{A}^{(ro, wo, \widehat{v}_0)} = 1 \right] - \Pr_{(ro, wo, \widehat{v}_0) \leftarrow \mathcal{O}(\lambda)} \left[ \mathcal{A}^{\mathcal{M}(ro, wo)} = 1 \right] \right| \leq \frac{t}{2^\lambda} . \quad (1)$$

<sup>6</sup> Emulators are sometimes known as “lazy samplers” or “simulators”. In this paper we reserve the word *simulator* to refer to zero knowledge simulators.

Moreover,  $\mathcal{M}$  is **degenerate** with respect to  $(ro, wo)$ : it answers queries to those oracles by forwarding them to the corresponding “real” oracle (and recording the answers).

We refer to the absolute difference in Eq. 1 as the *emulation error*. An emulator is *perfect* if it has zero emulation error.

**Prior Oracle Emulators.** Recall that a random oracle is a function  $ro$  chosen uniformly from  $(\{0, 1\}^m \rightarrow \{0, 1\}^\lambda)$ . It has a well-known perfect (stateful) emulator  $\mathcal{M}_{ro}$  that “lazily” samples answers: when  $\mathcal{M}_{ro}$  receives a new query  $x \in \{0, 1\}^m$ , it uniformly samples and returns  $y \in \{0, 1\}^\lambda$ , and then saves the query-answer pair  $(x, y)$  into its state; when  $\mathcal{M}_{ro}$  receives a repeat query, it returns the saved answer.

The low-degree random oracle [CCS22] also has a perfect emulator, based on *succinct constraint detection* for the Reed–Muller code [BCF+17].

**Challenges for the ARO.** The low-degree structure of  $\widehat{vo}$  may suggest that succinct constraint detection directly yields a construction of  $\mathcal{M}^{(ro,wo)}$  with perfect emulation. However, the “sparsity” of  $vo$  implies that the set of all possible  $\widehat{vo}$  is *not* a linear space, as we now explain. Recall that for  $x \in \{0, 1\}^m, y \in \{0, 1\}^\lambda, z \in \{0, 1\}^w$ ,  $\widehat{vo}(x, y, z) = vo(x, y, z) = 1$  if and only if  $y = ro(x)$  and  $z = wo(x)$ , and 0 otherwise. Hence, if  $\widehat{vo}_1, \widehat{vo}_2$  are extended verification oracles,  $\widehat{vo}' = \widehat{vo}_1 + \widehat{vo}_2$  may not be an extended verification oracle because there may exist  $x, y_1, y_2, z_1, z_2$  such that  $\widehat{vo}'(x, y_1, z_1) = \widehat{vo}'(x, y_2, z_2) = 1$  and  $y_1 \neq y_2$ . Hence, unlike for the LDRO, we cannot directly construct  $\mathcal{M}^{(ro,wo)}$  from succinct constraint detection.

**Our Approach.** We adopt a novel approach to simulation. First, we design a query-efficient but time-inefficient perfect emulator for a random low-degree extension  $\widehat{f}$  of a given arbitrary function  $f$ .<sup>7</sup> This *almost* suffices for our goal because  $\widehat{vo}$  is a random low-degree extension of the function  $vo$  defined by  $(ro, wo)$ , which we can efficiently compute at any point by querying  $ro$  and  $wo$ . Second, we additionally achieve time-efficient emulation by leveraging the *sparsity* of  $vo$ , at the cost of a small statistical emulation error.

**(1) Time-inefficient emulation of a random low-degree extension.** Let  $f : \{0, 1\}^n \rightarrow \mathbb{F}$  be a function and  $d \in \mathbb{N}$  a degree bound. We seek an emulator  $\mathcal{M}_{LD}^f$  such that  $\mathcal{M}_{LD}^f$  answers queries in a way that is identically distributed to a random extension  $\widehat{f}$  of  $f$  with individual degree at most  $d$ .

We fix some notation. For  $S \subseteq \mathbb{F}^n$  and  $w \in \{0, 1\}^n$ , we say that  $w$  is  $S$ -good if there exists an  $n$ -variate polynomial  $Q_{w,S}$  of individual degree at most  $d$  such that: (i)  $Q_{w,S}(w) = 1$ ; (ii)  $Q_{w,S}(x) = 0$  for every  $x \in \{0, 1\}^n \setminus \{w\}$ ; and (iii)  $Q_{w,S}(z) = 0$  for every  $z \in S$ . We say that  $w$  is  $S$ -bad if it is not  $S$ -good. Intuitively,  $w$  is  $S$ -bad if  $f(w)$  can be deduced from  $\widehat{f}|_S$  (given the structure of a low-degree extension  $\widehat{f}$ ). Note that  $S$ -badness is monotone with respect to  $S$ , and that if  $w \in S$  then  $w$  is  $S$ -bad.

The query-efficient but time-inefficient emulator  $\mathcal{M}_{LD}$  works as follows.

$\mathcal{M}_{LD}^f$ :

<sup>7</sup> In contrast, emulating the low-degree random oracle (as in [CCS22]) corresponds to emulating  $\widehat{f}$  for a *random* function  $f$  that the emulator *samples itself*. This considerably simplifies the task, and in particular enables a time-efficient perfect emulation.

1. Initially sample a random low-degree extension  $P$  of the all-zero function on  $\{0, 1\}^n$ .
2. For each new query  $x$ , answer it as follows.
  - Let  $S \subseteq \mathbb{F}^m$  be the set of points queried prior to  $x$ .
  - Let  $W$  denote the set of  $w \in \{0, 1\}^n$  that are  $S$ -good and  $(S \cup \{x\})$ -bad.
  - Update  $P := P + \sum_{w \in W} f(w) \cdot Q_{w,S}$ .
  - Return  $P(x)$  as the answer.

The emulator  $\mathcal{M}_{LD}$  maintains the invariant that  $P$  is a low-degree extension of the function  $g: \{0, 1\}^n \rightarrow \mathbb{F}$  given by  $g(w) = 0$  for  $S$ -good  $w$  and  $g(w) = f(w)$  for  $S$ -bad  $w$ . That is,  $g$  is consistent with  $f$  at every point the adversary “knows”, and is zero elsewhere. It is also crucial that  $\mathcal{M}_{LD}$  does not change  $P(x)$  for any  $x \in S$ , since such a change would be detectable by the adversary; this is achieved since  $Q_{w,S}(x) = 0$  for all  $x \in S$ . Together these facts imply that  $\mathcal{M}_{LD}$  achieves perfect simulation.

The query complexity of  $\mathcal{M}_{LD}$  is equal to the size of the union of all sets  $W$  across all invocations of Step 2, which is equal to the number of  $S$ -bad points when  $S$  is the set of all queries made to  $\hat{f}$ . Aaronson and Wigderson [AW09, Lemma 4.3] proved that, provided  $d \geq 2$ , the number of  $S$ -bad points is at most  $|S|$ , which in the context of Lemma 2 is the query complexity of  $\mathcal{A}$ .

**(2) Time-efficient emulation from sparsity.** There are three main sources of time-inefficiency in the emulator  $\mathcal{M}_{LD}$ : (i) sampling the initial polynomial  $P$ ; (ii) computing the polynomials  $Q_{w,S}$ ; and (iii) computing the set  $W$ . We consider each of these difficulties in turn. Throughout we will make use of the random multivariate polynomial emulation algorithm of [BCF+17], which achieves the following guarantee.

**Lemma 3.** *There is an efficient probabilistic algorithm  $\text{LDSample}$  such that for every degree bound  $d \in \mathbb{N}$ , set  $S \subseteq \mathbb{F}^m$ , map  $h: S \rightarrow \mathbb{F}$ ,  $q \in \mathbb{F}^m$ , and  $\alpha \in \mathbb{F}$ ,*

$$\Pr[\text{LDSample}(1^d, S, h, q) = \alpha] = \Pr[P(q) = \alpha \mid P|_S = h] ,$$

where  $P$  is a uniformly random  $m$ -variate polynomial of individual degree at most  $d$ .<sup>8</sup>

We address the first two difficulties via suitable use of algebra.

- (i) *Sampling  $P$ .* The polynomial  $P$  is initially a random low-degree extension of the all-zero function. We do not know how to use  $\text{LDSample}$  to sample  $P$  directly, since that would need  $S = \{0, 1\}^m$ , which is an exponentially-sized set. Instead, we use a structural result about low-degree extensions of the zero function, the *combinatorial nullstellensatz* [Alo99].

**Lemma 4 (informal).** *If a polynomial  $P$  is zero on  $\{0, 1\}^m$ , then there exist polynomials  $(R_i)_{i=1}^m$  such that*

$$P(\mathbf{X}) \equiv \sum_{i=1}^m X_i(X_i - 1)R_i(\mathbf{X}) . \tag{2}$$

<sup>8</sup> If the RHS is not well-defined,  $\text{LDSample}$  outputs  $\perp$ .

Combining Lemma 4 with a linear-algebraic argument, we show that sampling each  $R_i$  in Eq. 2 uniformly at random yields a uniformly random low-degree extension of the all-zero function. We can then sample each  $R_i$  via LDSample.

- (ii) *Computing  $Q_{w,S}$ .* [AW09] sets the polynomial  $Q_{w,S} := \delta_w \cdot p_{w,S}$  where:
  - $\delta_w$  is the unique multilinear polynomial with  $\delta_w(w) = 1$  and  $\delta_w(x) = 0$  for all  $x \in \{0, 1\}^m \setminus \{w\}$ ;
  - $p_{w,S}$  is a multilinear polynomial with  $p_{w,S}(w) = 1$  and  $p_{w,S}(z) = 0$  for all  $z \in S$ .

The polynomial  $\delta_w$  is easy to compute because it has a succinct expression. In contrast, the polynomial  $p_{w,S}$  may not have a succinct expression, but is specified via its evaluations on the polynomial-sized set  $S \cup \{w\}$ , so queries to  $p_{w,S}$  can be answered via LDSample.

We do not know of an algorithm that can efficiently compute, given a set  $S \subseteq \mathbb{F}^n$ , the set of all  $S$ -bad points. As a result, we do not know how to obtain an efficient emulator for a random low-degree extension of an arbitrary function  $f$ . Nevertheless we address the third difficulty by leveraging the structure of  $\text{vo}$ .

- (iii) *The set  $W$ .* We observe that  $\text{vo}$  is sparse: it is nonzero only at points  $(x, y, z)$  for  $\text{ro}(x) = y, \text{wo}(x) = z$ . If the adversary has not queried  $\text{ro}$  at  $x$ , then intuitively (since  $\text{ro}(x)$  is random) it will not be able to find any  $y, z$  such that  $\text{vo}(x, y, z) = 1$ , even given access to  $\widehat{\text{vo}}$ . In particular, the probability that the set  $W$  contains any  $(x, y, z) \in \{0, 1\}^{m+\lambda+w}$  such that  $\text{vo}(x, y, z) = 1$ , but  $\text{ro}(x)$  was not yet queried, should be negligible. Indeed, we show that this probability is at most  $|S|/2^\lambda$ .

Observe that Step 2 of the time-inefficient emulator does nothing if  $f(w) = 0$  for all  $w \in W$ . It follows from the above that to achieve simulation accuracy  $O(|S|/2^\lambda)$ , it suffices to perform Step 2 only for points  $(x, y, z)$  for which the adversary has already queried  $\text{ro}$  at  $x$  and  $\text{ro}(x) = y, \text{wo}(x) = z$ . Since we observe the adversary’s queries to  $\text{ro}$ , this set of points is easy to determine.

To show this formally we follow an “identical-until-bad-is-set” analysis [BR06].

### 3 Preliminaries

#### 3.1 Notations

We define  $[n] := \{1, \dots, n\}$ . We use  $\mathbb{F}^{\leq d}[X_1, \dots, X_m]$  to denote the set of  $m$ -variate polynomials of individual degree at most  $d$  with coefficients in  $\mathbb{F}$ ; we write  $\text{deg}(\cdot)$  to denote the individual degree. For  $\mathbf{d} = (d_1, \dots, d_m)$ , we use  $\mathbb{F}^{\leq \mathbf{d}}[X_1, \dots, X_m]$  to denote the set of  $m$ -variate polynomials such that the variable  $X_i$  has individual degree at most  $d_i$  for each  $i \in [m]$ .

**Functions.** We use  $\text{Dom}(f)$  to denote the domain and  $\text{Cod}(f)$  to denote the codomain of a function  $f$ . We use  $(X \rightarrow Y)$  to denote the set of all functions  $\{f: X \rightarrow Y\}$ . For a linear map  $\Phi$ , we use  $\ker(\Phi)$  to denote the kernel of  $\Phi$  and  $\text{im}(\Phi)$  to denote the image of  $\Phi$ . We say that a function is total if it is defined for all elements of its domain, and say that it is not total otherwise.

**Distributions.** For finite set  $X$ , we write  $x \leftarrow X$  to denote that  $x$  is drawn uniformly at random from  $X$ . We use  $\text{supp}(\mathcal{D})$  to denote the support of the distribution  $\mathcal{D}$ . We write  $\mathcal{U}(X)$  to denote the uniform distribution over the set  $X$ .

**Oracles.** A *random oracle* is defined as a function  $\text{ro}$  sampled uniformly at random from  $(\{0, 1\}^m \rightarrow \{0, 1\}^n)$  for some  $m, n \in \mathbb{N}$ .

**Oracle Algorithms.** For a function  $\theta: X \rightarrow Y$ , we write  $A^\theta$  for an algorithm with oracle access to  $\theta$ . Further, for a tuple of functions  $(\theta_1, \dots, \theta_\nu)$ , where  $\nu \in \mathbb{N}$ , with  $\theta_i: X_i \rightarrow Y_i$ , we write  $A^{(\theta_1, \dots, \theta_\nu)}$  for an algorithm with oracle access to each  $\theta_i$  for  $i \in [\nu]$ , and  $A^{\theta_S}$  for an algorithm with oracle access to a subset of functions  $\{\theta_i \mid i \in S\}$  where  $S \subseteq [\nu]$ .

**Oracle Identifiers and Oracle Transcripts.** Given an oracle distribution  $\mathcal{O}$  such that  $\text{supp}(\mathcal{O})$  contains tuples of  $\nu \in \mathbb{N}$  oracles, we assign to each oracle in the tuple a unique oracle identifier:  $\text{oid} \in [\nu]$ . Then, an  $\mathcal{O}$ -query-answer transcript  $\text{tr}$  is a list consisting of query-answer pairs, along with the oracle identifier corresponding to the oracle to which each query was made. That is,  $\text{tr} := [(\text{oid}_i, x_i, y_i)]_{i=1}^t$  such that there exists  $(\theta_1, \theta_2, \dots, \theta_\nu) \in \text{supp}(\mathcal{O})$  satisfying  $\theta_{\text{oid}_i}(x_i) = y_i$  for all  $i \in [t]$ . (Note that  $\text{oid}_i \in [\nu]$  indicates that the  $i$ -th query was made to  $\theta_{\text{oid}_i}$ .)

When considering oracle distributions  $\mathcal{O}$  whose support contains tuples of oracles, it is often useful to view this tuple  $(\theta_1, \dots, \theta_\nu)$  as a bundle of oracles  $\theta: [\nu] \times \bigcup_{(\theta_1, \dots, \theta_\nu) \in \text{supp}(\mathcal{O})} \bigcup_{i \in [\nu]} \text{Dom}(\theta_i) \rightarrow \bigcup_{(\theta_1, \dots, \theta_\nu) \in \text{supp}(\mathcal{O})} \bigcup_{i \in [\nu]} \text{Cod}(\theta_i)$  which takes an oracle identifier as input alongside the query point, i.e.,  $\theta \leftarrow \mathcal{O}$  and  $\theta(\text{oid}_i, x) = \theta_{\text{oid}_i}(x)$ .

**Query Complexity.** An algorithm with access to an oracle  $\mathcal{O}$  is  $t$ -query if its  $\mathcal{O}$ -query-answer transcript has length  $\leq t$ .

**Indexed Relations.** An *indexed relation*  $\mathcal{R}$  is a set of triples  $(\mathfrak{i}, \mathfrak{x}, \mathfrak{w})$  where  $\mathfrak{i}$  is the index,  $\mathfrak{x}$  is the instance, and  $\mathfrak{w}$  is the witness; the corresponding *indexed language*  $\mathcal{L}(\mathcal{R})$  is the set of index-instance pairs  $(\mathfrak{i}, \mathfrak{x})$  for which there exists a witness  $\mathfrak{w}$  such that  $(\mathfrak{i}, \mathfrak{x}, \mathfrak{w}) \in \mathcal{R}$ . For example, the indexed relation of satisfiable boolean circuits consists of triples where  $\mathfrak{i}$  is the description of a boolean circuit,  $\mathfrak{x}$  is a partial assignment to its input wires, and  $\mathfrak{w}$  is an assignment to the remaining wires that makes the circuit output 0.

**Oracle Relations.** For a set of oracle distributions  $\mathcal{X}$ , we write  $\mathcal{R}^\mathcal{X}$  to denote the set of indexed relations  $\{\mathcal{R}^\theta : \theta \in \bigcup_{\mathcal{O} \in \mathcal{X}} \text{supp}(\mathcal{O})\}$ . When considering sets of oracle distributions  $\mathcal{X}$  for which each  $\mathcal{O} \in \mathcal{X}$  is such that  $\text{supp}(\mathcal{O})$  contains tuples of oracles  $(\theta_1, \dots, \theta_\nu)$ , for  $\nu \in \mathbb{N}$ , with oracle identifiers  $(\text{oid}_1, \dots, \text{oid}_\nu)$ , we write  $\mathcal{R}^{(\mathcal{X}, \text{oid})}$  to denote the set of indexed relations  $\{\mathcal{R}^{\theta_{\text{oid}}} : (\theta_1, \dots, \theta_\nu) \in \bigcup_{\mathcal{O} \in \mathcal{X}} \text{supp}(\mathcal{O})\}$ . We define  $\mathcal{R}^{(\mathcal{X}, \text{oid})} \in \text{NP}^{(\mathcal{X}, \text{oid})}$  if and only if there exists a polynomial-time oracle Turing machine  $M$  such that, for every  $(\theta_1, \dots, \theta_\nu) \in \bigcup_{\mathcal{O} \in \mathcal{X}} \text{supp}(\mathcal{O})$ ,  $\mathcal{R}^{\theta_{\text{oid}}} = \{(\mathfrak{i}, \mathfrak{x}, \mathfrak{w}) : M^{\theta_{\text{oid}}}(\mathfrak{i}, \mathfrak{x}, \mathfrak{w}) = 1\}$ .

**Security Parameters.** We assume for simplicity that all public parameters have a length of at least  $\lambda$  so that efficient algorithms that receive such parameters can run in time (at least) polynomial in  $\lambda$ .

**Adversaries.** An adversary (or extractor) is *polynomial-size* if it can be expressed as a circuit of polynomial size. We also consider a relaxed definition: an adversary (or extractor) running in (*non-uniform*) *expected polynomial-time* is a Turing machine provided with a *polynomial-size* non-uniform advice string and access to an infinite random tape, whose expected running time for all choices of advice is polynomial.

An adversary  $\mathcal{A}$  with expected running time  $t$  and success probability  $p$  can be converted into a circuit of size  $O(t/\epsilon)$  with success probability  $p - \epsilon$  as follows: first truncate the execution of  $\mathcal{A}$  at running time  $t/\epsilon$ ; then choose as advice the randomness that maximizes the success probability of the truncated  $\mathcal{A}$ .

For  $\nu \in \mathbb{N}$  and a distribution  $\mathcal{O}$ , whose support contains tuples of oracles  $(\theta_1, \dots, \theta_\nu)$ , we refer to an adversary with access to  $(\theta_1, \dots, \theta_\nu) \leftarrow \mathcal{O}$  as an  $\mathcal{O}$ -adversary.

**Stateful Algorithms.** An algorithm  $A$  is *stateful* if it has the following syntax:

- $A.\text{Initialize}(\text{pp}) \rightarrow z$ , on input parameters  $\text{pp}$ , outputs an initial state  $z$ .
- $A.\text{Evaluate}(\text{pp}, z_0, x) \rightarrow (z_1, y)$ , on input an old state  $z_0$  and a query  $x$ , outputs a new state  $z_1$  and an output  $y$ .

### 3.2 Non-interactive Arguments in Oracle Models

Given a set of oracle distributions  $\mathcal{X}$ , a (preprocessing) *non-interactive argument* relative for an indexed oracle relation  $\mathcal{R}^{\mathcal{X}}$  is a tuple of algorithms  $\text{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$  that works as follows. Below we denote by  $\theta$  an oracle (or tuple of oracles) in the set  $\bigcup_{\mathcal{O} \in \mathcal{X}} \text{supp}(\mathcal{O})$ .

- $\mathcal{G}(1^\lambda) \rightarrow \text{pp}$ . On input a security parameter  $\lambda$  (in unary), the generator  $\mathcal{G}$  samples public parameters  $\text{pp}$ .
- $\mathcal{I}^\theta(\text{pp}, \mathfrak{i}) \rightarrow (\text{ipk}, \text{ivk})$ . On input public parameters  $\text{pp}$  and an index  $\mathfrak{i}$  for the relation  $\mathcal{R}$ , the indexer  $\mathcal{I}$  deterministically computes index-specific proving and verification keys  $(\text{ipk}, \text{ivk})$ .
- $\mathcal{P}^\theta(\text{ipk}, \mathfrak{x}, \mathfrak{w}) \rightarrow \pi$ . On input an index-specific proving key  $\text{ipk}$ , an instance  $\mathfrak{x}$ , and a corresponding witness  $\mathfrak{w}$ , the prover  $\mathcal{P}$  computes a proof  $\pi$  that attests to the claim that  $(\mathfrak{i}, \mathfrak{x}, \mathfrak{w}) \in \mathcal{R}^\theta$ .
- $\mathcal{V}^\theta(\text{ivk}, \mathfrak{x}, \pi) \rightarrow b$ . On input an index-specific verification key  $\text{ivk}$ , an instance  $\mathfrak{x}$ , and a corresponding proof  $\pi$ , the verifier  $\mathcal{V}$  computes a bit indicating whether  $\pi$  is a valid proof.

We require ARG to satisfy the following completeness and soundness properties.

- *Completeness.* For every oracle distribution  $\mathcal{O} \in \mathcal{X}$  and adversary  $\mathcal{A}$ ,

$$\Pr \left[ \begin{array}{c|c} (\mathfrak{i}, \mathfrak{x}, \mathfrak{w}) \in \mathcal{R}^\theta & \theta \leftarrow \mathcal{O}(\lambda) \\ \downarrow & \text{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \mathcal{V}^\theta(\text{ivk}, \mathfrak{x}, \pi) = 1 & \begin{array}{l} (\mathfrak{i}, \mathfrak{x}, \mathfrak{w}) \leftarrow \mathcal{A}^\theta(\text{pp}) \\ (\text{ipk}, \text{ivk}) \leftarrow \mathcal{I}^\theta(\text{pp}, \mathfrak{i}) \\ \pi \leftarrow \mathcal{P}^\theta(\text{ipk}, \mathfrak{x}, \mathfrak{w}) \end{array} \end{array} \right] = 1 .$$

The above formulation of completeness allows  $(\mathfrak{i}, \mathfrak{x}, \mathfrak{w})$  to depend on the oracle  $\theta$  and public parameters  $\text{pp}$ .



– *Soundness*. For every oracle distribution  $\mathcal{O} \in \mathcal{X}$  and polynomial-size adversary  $\tilde{\mathcal{P}}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}^\theta(\text{ivk}, \mathbb{x}, \pi) = 1 \\ \wedge \\ (\hat{\mathbf{i}}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}^\theta) \end{array} \middle| \begin{array}{l} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\hat{\mathbf{i}}, \mathbb{x}, \pi) \leftarrow \tilde{\mathcal{P}}^\theta(\text{pp}) \\ (\text{ipk}, \text{ivk}) \leftarrow \mathcal{I}^\theta(\text{pp}, \hat{\mathbf{i}}) \end{array} \right] \leq \text{negl}(\lambda) .$$

The above formulation of soundness allows  $(\hat{\mathbf{i}}, \mathbb{x})$  to depend on the oracle  $\theta$  and public parameters  $\text{pp}$ .

We also consider straightline knowledge soundness properties and zero knowledge for ARG.

**Straightline Knowledge Soundness.** ARG has *straightline knowledge soundness* (with respect to auxiliary input distribution  $\mathcal{D}$ ) if there exists a deterministic polynomial-time extractor  $\mathcal{E}$  such that for every oracle distribution  $\mathcal{O} \in \mathcal{X}$  and (non-uniform) polynomial-time adversary  $\tilde{\mathcal{P}}$ ,

$$\Pr \left[ \begin{array}{l} \mathcal{V}^\theta(\text{ivk}, \mathbb{x}, \pi) = 1 \\ \wedge \\ (\hat{\mathbf{i}}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}^\theta \end{array} \middle| \begin{array}{l} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \text{ai} \leftarrow \mathcal{D}(\text{pp}) \\ (\hat{\mathbf{i}}, \mathbb{x}, \pi) \stackrel{\text{tr}}{\leftarrow} \tilde{\mathcal{P}}^\theta(\text{pp}, \text{ai}) \\ (\text{ipk}, \text{ivk}) \leftarrow \mathcal{I}^\theta(\text{pp}, \hat{\mathbf{i}}) \\ \mathbb{w} \leftarrow \mathcal{E}(\text{pp}, \hat{\mathbf{i}}, \mathbb{x}, \pi, \text{tr}) \end{array} \right] \leq \text{negl}(\lambda) .$$

**Zero Knowledge.** ARG has statistical zero knowledge if there exists a probabilistic polynomial-time stateful simulator  $\mathcal{S}$  such that for every oracle distribution  $\mathcal{O} \in \mathcal{X}$  and polynomial-size honest stateful adversary  $\mathcal{A}$ , the following distributions are  $\text{negl}(\lambda)$ -close in statistical distance:

$$\left\{ \mathcal{A}^\theta(\pi) \middle| \begin{array}{l} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\hat{\mathbf{i}}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^\theta(\text{pp}) \\ (\text{ipk}, \text{ivk}) \leftarrow \mathcal{I}^\theta(\text{pp}, \hat{\mathbf{i}}) \\ \pi \leftarrow \mathcal{P}^\theta(\text{ipk}, \mathbb{x}, \mathbb{w}) \end{array} \right\} \text{ and } \left\{ \mathcal{A}^{\mathcal{S}^\theta}(\pi) \middle| \begin{array}{l} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathcal{S}(1^\lambda) \\ (\hat{\mathbf{i}}, \mathbb{x}, \mathbb{w}) \stackrel{\text{tr}}{\leftarrow} \mathcal{A}^\theta(\text{pp}) \\ \pi \leftarrow \mathcal{S}^\theta(\hat{\mathbf{i}}, \mathbb{x}, \text{tr}) \end{array} \right\} . \quad (3)$$

An adversary  $\mathcal{A}$  is *honest* if it outputs  $(\hat{\mathbf{i}}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}^\theta$  with probability 1. Above, the notation  $\mathcal{A}^{\mathcal{S}^\theta}$  indicates that the simulator  $\mathcal{S}$  (with oracle access to  $\theta$ ) answers the oracle queries of  $\mathcal{A}$ .

**Succinctness.** In this work, we say that a non-interactive argument system ARG for  $\mathcal{R}^\mathcal{X}$  is *succinct* if there is a fixed polynomial  $p$  such that *both* the length of the proof and the running time of the argument verifier are bounded by  $p(\lambda, |\mathbb{x}|)$ . In this case, we refer to ARG as a SNARG; if ARG also has knowledge soundness, it is a SNARK.

### 3.3 Proof-Carrying Data

A triple of algorithms  $\text{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$  is a (preprocessing) *proof-carrying data scheme* (PCD scheme) for a class of compliance predicates  $\mathbb{F}$  relative to a set of oracle distributions  $\mathcal{X}$  if the properties below hold.

**Definition 1.** A **transcript**  $\mathsf{T}$  is a directed acyclic graph where each vertex  $u \in V(\mathsf{T})$  is labeled by local data  $z_{\text{loc}}^{(u)}$  and each edge  $e \in E(\mathsf{T})$  is labeled by a message  $z^{(e)} \neq \perp$ . The **output** of a transcript  $\mathsf{T}$ , denoted  $\text{o}(\mathsf{T})$ , is  $z^{(e)}$  where  $e = (u, v)$  is the lexicographically-first edge such that  $v$  is a sink.

**Definition 2.** A vertex  $u \in V(\mathsf{T})$  is  $\Phi$ -compliant for  $\Phi \in \mathsf{F}$  if for all outgoing edges  $e = (u, v) \in E(\mathsf{T})$  and for all  $\theta \in \bigcup_{\mathcal{O} \in \mathcal{X}} \text{supp}(\mathcal{O})$ :

- (base case) if  $u$  has no incoming edges,  $\Phi^\theta(z^{(e)}, z_{\text{loc}}^{(u)}, \perp, \dots, \perp) = 1$ ;
- (recursive case) if  $u$  has incoming edges  $e_1, \dots, e_m$ ,  $\Phi^\theta(z^{(e)}, z_{\text{loc}}^{(u)}, z^{(e_1)}, \dots, z^{(e_m)}) = 1$ .

We say that  $\mathsf{T}$  is  $\Phi$ -compliant if  $E(\mathsf{T})$  is non-empty and all vertices incident to an edge are  $\Phi$ -compliant.

**Completeness.** For every oracle distribution  $\mathcal{O} \in \mathcal{X}$  and adversary  $\mathcal{A}$ ,

$$\Pr \left[ \left( \begin{array}{c} \Phi \in \mathsf{F} \\ \wedge \left( (\wedge_{i=1}^m z_i = \perp) \vee (\wedge_{i=1}^m \mathbb{V}^\theta(\text{ivk}, z_i, \pi_i) = 1) \right) \\ \wedge \Phi^\theta(z, z_{\text{loc}}, z_1, \dots, z_m) = 1 \\ \downarrow \\ \mathbb{V}^\theta(\text{ivk}, z, \pi) = 1 \end{array} \right) \middle| \begin{array}{c} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\Phi, z, z_{\text{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}^\theta(\text{pp}) \\ (\text{ipk}, \text{ivk}) \leftarrow \mathbb{I}^\theta(\text{pp}, \Phi) \\ \pi \leftarrow \mathbb{P}^\theta(\text{ipk}, z, z_{\text{loc}}, [z_i, \pi_i]_{i=1}^m) \end{array} \right] = 1 .$$

**Straightline Knowledge Soundness.**  $\text{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$  has straightline knowledge soundness (with respect to auxiliary input distribution  $\mathcal{D}$ ) if there exists a deterministic polynomial-time extractor  $\mathbb{E}$  such that for every oracle distribution  $\mathcal{O} \in \mathcal{X}$  and (non-uniform) polynomial-time adversary  $\tilde{\mathbb{P}}$ ,

$$\Pr \left[ \begin{array}{c} \Phi \in \mathsf{F} \\ \wedge \mathbb{V}(\text{ivk}, \text{o}, \pi) = 1 \\ \wedge \left( \mathsf{T} \text{ is not } \Phi\text{-compliant} \vee \text{o}(\mathsf{T}) \neq \text{o} \right) \end{array} \middle| \begin{array}{c} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \text{ai} \leftarrow \mathcal{D}(\text{pp}) \\ (\Phi, \text{o}, \pi) \leftarrow \text{tr } \tilde{\mathbb{P}}^\theta(\text{pp}, \text{ai}) \\ (\text{ipk}, \text{ivk}) \leftarrow \mathbb{I}^\theta(\text{pp}, \Phi) \\ \mathsf{T} \leftarrow \mathbb{E}(\text{pp}, \Phi, \text{o}, \pi, \text{tr}) \end{array} \right] \leq \text{negl}(\lambda) .$$

**Zero Knowledge.**  $\text{PCD}$  has statistical zero knowledge if there exists a probabilistic polynomial-time stateful simulator  $\mathbb{S}$  such that for every oracle distribution  $\mathcal{O} \in \mathcal{X}$  and polynomial-size honest (stateful) adversary  $\mathcal{A}$ , the following distributions are  $\text{negl}(\lambda)$ -close in statistical distance:

$$\left\{ \mathcal{A}^\theta(\pi) \middle| \begin{array}{c} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\Phi, z, z_{\text{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}^\theta(\text{pp}) \\ (\text{ipk}, \text{ivk}) \leftarrow \mathbb{I}^\theta(\text{pp}, \Phi) \\ \pi \leftarrow \mathbb{P}^\theta(\text{ipk}, \Phi, z, z_{\text{loc}}, [z_i, \pi_i]_{i=1}^m) \end{array} \right\} \text{ and } \left\{ \mathcal{A}^{\mathbb{S}^\theta}(\pi) \middle| \begin{array}{c} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathbb{S}(1^\lambda) \\ (\Phi, z, z_{\text{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \text{tr } \mathcal{A}^\theta(\text{pp}) \\ \pi \leftarrow \mathbb{S}^\theta(\Phi, z, \text{tr}) \end{array} \right\} .$$

An adversary  $\mathcal{A}$  is *honest* if its output satisfies the implicant of the completeness condition with probability 1 (i.e.,  $\Phi \in \mathbb{F}$ ,  $\Phi^\theta(z, z_{\text{loc}}, z_1, \dots, z_m) = 1$ , and either for all  $i$ ,  $z_i = \perp$ , or for all  $i$ ,  $\mathbb{V}^\theta(\text{ivk}, z_i, \pi_i) = 1$ ). Above, the notation  $\mathcal{A}^{\mathbb{S}}$  indicates that the simulator  $\mathbb{S}$  answers oracle queries of  $\mathcal{A}$ .

**Efficiency.** The generator  $\mathbb{G}$ , prover  $\mathbb{P}$ , indexer  $\mathbb{I}$  and verifier  $\mathbb{V}$  run in polynomial time. A proof  $\pi$  has size  $\text{poly}(\lambda, |\Phi|)$ ; in particular, it does not grow with each application of  $\mathbb{P}$ .

### 3.4 Accumulation Schemes

We recall the definition of an accumulation scheme from [BCMS20], extended to any set of oracle distributions; then, in Definition 3 below, we describe how to specialize that notion to the case of accumulating oracle queries.

Let  $\Phi: \bigcup_{\mathcal{O} \in \mathcal{X}} \text{supp}(\mathcal{O}(\cdot)) \times (\{0, 1\}^*)^3 \rightarrow \{0, 1\}$  be a predicate (for clarity we write  $\Phi^\theta(\text{pp}_\Phi, \text{i}_\Phi, \text{q})$  for  $\Phi(\theta, \text{pp}_\Phi, \text{i}_\Phi, \text{q})$ ). Let  $\mathcal{H}$  be a probabilistic algorithm with access to  $\theta$ , which outputs predicate parameters  $\text{pp}_\Phi$ .

An **accumulation scheme for**  $(\Phi, \mathcal{H})$  is a tuple of algorithms  $\text{AS} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V}, \mathbb{D})$  that have access to the same oracle  $\theta$  (except for  $\mathbb{G}$ ). These algorithms satisfy *completeness* and *soundness*, and optionally also *zero knowledge*, as specified below.

**Completeness.** For every oracle distribution  $\mathcal{O} \in \mathcal{X}$  and (unbounded) adversary  $\mathcal{A}$ ,

$$\Pr \left[ \begin{array}{l} \forall j \in [\ell], \mathbb{D}^\theta(\text{dk}, \text{acc}_j) = 1 \\ \forall i \in [n], \Phi^\theta(\text{pp}_\Phi, \text{i}_\Phi, \text{q}_i) = 1 \\ \Downarrow \\ \mathbb{V}^\theta(\text{avk}, [\text{q}_i]_{i=1}^n, [\text{acc}_j]_{j=1}^\ell, \text{acc}, \pi_V) = 1 \\ \mathbb{D}^\theta(\text{dk}, \text{acc}) = 1 \end{array} \middle| \begin{array}{l} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \text{pp}_\Phi \leftarrow \mathcal{H}^\theta(1^\lambda) \\ (\text{i}_\Phi, [\text{q}_i]_{i=1}^n, [\text{acc}_j]_{j=1}^\ell) \leftarrow \mathcal{A}^\theta(\text{pp}, \text{pp}_\Phi) \\ (\text{apk}, \text{avk}, \text{dk}) \leftarrow \mathbb{I}^\theta(\text{pp}, \text{pp}_\Phi, \text{i}_\Phi) \\ (\text{acc}, \pi_V) \leftarrow \mathbb{P}^\theta(\text{apk}, [\text{q}_i]_{i=1}^n, [\text{acc}_j]_{j=1}^\ell) \end{array} \right] = 1 .$$

Note that for  $\ell = n = 0$ , the precondition on the left-hand side holds vacuously; this is required for the completeness condition to be non-trivial.

**Soundness.** For every oracle distribution  $\mathcal{O} \in \mathcal{X}$  and polynomial-size adversary  $\mathcal{A}$ ,

$$\Pr \left[ \begin{array}{l} \mathbb{V}^\theta(\text{avk}, [\text{q}_i]_{i=1}^n, [\text{acc}_j]_{j=1}^\ell, \text{acc}, \pi_V) = 1 \\ \mathbb{D}^\theta(\text{dk}, \text{acc}) = 1 \\ \Downarrow \\ \forall j \in [\ell], \mathbb{D}^\theta(\text{dk}, \text{acc}_j) = 1 \\ \forall i \in [n], \Phi^\theta(\text{pp}_\Phi, \text{i}_\Phi, \text{q}_i) = 1 \end{array} \middle| \begin{array}{l} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \text{pp}_\Phi \leftarrow \mathcal{H}^\theta(1^\lambda) \\ (\text{i}_\Phi, [\text{q}_i]_{i=1}^n, [\text{acc}_j]_{j=1}^\ell) \leftarrow \mathcal{A}^\theta(\text{pp}, \text{pp}_\Phi) \\ \text{acc} \leftarrow \pi_V \\ (\text{apk}, \text{avk}, \text{dk}) \leftarrow \mathbb{I}^\theta(\text{pp}, \text{pp}_\Phi, \text{i}_\Phi) \end{array} \right] \geq 1 - \text{negl}(\lambda) .$$

**Zero Knowledge.** There exists a polynomial-time stateful simulator  $\mathbb{S}$  such that for every oracle distribution  $\mathcal{O} \in \mathcal{X}$  and polynomial-size stateful “honest” adversary  $\mathcal{A}$  (see below), the following distributions are (statistically/computationally) indistinguishable:

$$\left\{ \mathcal{A}^\theta(\text{acc}) \middle| \begin{array}{l} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \text{pp}_\Phi \leftarrow \mathcal{H}^\theta(1^\lambda) \\ (\text{i}_\Phi, [\text{q}_i]_{i=1}^n, [\text{acc}_j]_{j=1}^\ell) \leftarrow \mathcal{A}^\theta(\text{pp}, \text{pp}_\Phi) \\ (\text{apk}, \text{avk}, \text{dk}) \leftarrow \mathbb{I}^\theta(\text{pp}, \text{pp}_\Phi, \text{i}_\Phi) \\ (\text{acc}, \pi_V) \leftarrow \mathbb{P}^\theta(\text{apk}, [\text{q}_i]_{i=1}^n, [\text{acc}_j]_{j=1}^\ell) \end{array} \right\}$$

and

$$\left\{ \mathcal{A}^\theta(\text{acc}) \left| \begin{array}{l} \theta \leftarrow \mathcal{O}(\lambda) \\ \text{pp} \leftarrow \mathcal{S}(1^\lambda) \\ \text{pp}_{\mathcal{F}} \leftarrow \mathcal{H}^\theta(1^\lambda) \\ (i_\Phi, [q_i]_{i=1}^n, [\text{acc}_j]_{j=1}^\ell) \stackrel{\text{tr}}{\leftarrow} \mathcal{A}^\theta(\text{pp}, \text{pp}_{\mathcal{F}}) \\ \text{acc} \leftarrow \mathcal{S}^\theta(\text{pp}_{\mathcal{F}}, i_\Phi, \text{tr}) \end{array} \right. \right\} .$$

Here  $\mathcal{A}$  is *honest* if it outputs, with probability 1, a tuple  $(i_\Phi, [q_i]_{i=1}^n, [\text{acc}_j]_{j=1}^\ell)$  such that  $\Phi^\theta(\text{pp}_{\mathcal{F}}, i_\Phi, q_i) = 1$  and  $D^\theta(\text{dk}, \text{acc}_j) = 1$  for every  $i \in [n]$  and  $j \in [\ell]$ . Note that the simulator  $\mathcal{S}$  is *not* required to simulate the accumulation verifier proof  $\pi_V$ .

**Accumulation Scheme for Oracle Queries.** We explain how to specialize the general notion of an accumulation scheme above to the particular case of accumulating queries to a tuple of oracles.

**Definition 3.** Let  $\mathcal{X}$  be a set of oracle distributions. An **accumulation scheme for  $\mathcal{X}$ -queries** is an accumulation scheme where: (i) the accumulation verifier  $V$  does not access the oracle; (ii)  $\mathcal{H} = \perp$  (and so  $\text{pp}_{\mathcal{F}} = \perp$ ); (iii) predicate inputs  $q$  are of the form  $(x, y)$ ,<sup>9</sup> (iv) the predicate  $\Phi$  is defined such that  $\Phi^\theta(\text{pp}_{\mathcal{F}}, i_\Phi, x, y) = 1$  if and only if  $\theta(x) = y$  (in particular,  $\text{pp}_{\mathcal{F}}$  and  $i_\Phi$  are ignored).

### 3.5 Commitment Schemes

Let  $\nu \in \mathbb{N}$  and let  $\mathcal{X}$  be a set of oracle distributions, such that each  $\mathcal{O} \in \mathcal{X}$  is a distribution over tuples of oracles  $(\theta_1, \dots, \theta_\nu)$ . A *commitment scheme in  $\mathcal{X}$*  is a tuple  $\text{CM} = (\text{CM.Setup}, \text{CM.Commit})$  with the following syntax.

- **CM.Setup**, on input a security parameter  $1^\lambda$ , outputs a commitment key  $\text{ck}$ .
- **CM.Commit**, on input a commitment key  $\text{ck}$ , a message  $m \in \{0, 1\}^*$ , and randomness  $\omega$ , outputs a commitment  $\text{cm}$ .

The tuple  $\text{CM}$  satisfies a binding property and, optionally, a hiding property.

- **Binding.** For every  $\mathcal{O} \in \mathcal{X}$  and efficient adversary  $\mathcal{A}$ ,

$$\Pr \left[ \begin{array}{c} m_0 \neq m_1 \\ \wedge \\ \text{CM.Commit}(\text{ck}, m_0; \omega_0) = \text{CM.Commit}(\text{ck}, m_1; \omega_1) \end{array} \left| \begin{array}{l} (\theta_1, \dots, \theta_\nu) \leftarrow \mathcal{O}(\lambda) \\ \text{ck} \leftarrow \text{CM.Setup}(1^\lambda) \\ ((m_0, \omega_0), (m_1, \omega_1)) \leftarrow \mathcal{A}^{(\theta_1, \dots, \theta_\nu)}(\text{ck}) \end{array} \right. \right] \leq \text{negl}(\lambda) .$$

- **Hiding.** For every  $\mathcal{O} \in \mathcal{X}$  and efficient stateful adversary  $\mathcal{A}$  that outputs two messages of the same length, the following distributions are (statistically or computationally) indistinguishable:

$$\mathcal{D}_0(\lambda) := \left\{ (\text{pp}, \text{cm}, \text{aux}) \left| \begin{array}{l} (\theta_1, \dots, \theta_\nu) \leftarrow \mathcal{O}(\lambda) \\ \text{ck} \leftarrow \text{CM.Setup}(1^\lambda) \\ (m_0, m_1, \text{aux}) \leftarrow \mathcal{A}^{(\theta_1, \dots, \theta_\nu)}(\text{ck}) \\ \omega \leftarrow \{0, 1\}^{\text{poly}(\lambda)} \\ \text{cm} := \text{CM.Commit}(\text{ck}, m_0; \omega) \end{array} \right. \right\}$$

<sup>9</sup> If  $\mathcal{X}$  is a set of oracle distributions whose support contains tuples of oracles, then  $x$  is assumed to start with the oracle identifier corresponding to the oracle being queried.

$$\text{and } \mathcal{D}_1(\lambda) := \left\{ (\text{pp}, \text{cm}, \text{aux}) \left| \begin{array}{l} (\theta_1, \dots, \theta_\nu) \leftarrow \mathcal{O}(\lambda) \\ \text{ck} \leftarrow \text{CM.Setup}(1^\lambda) \\ (m_0, m_1, \text{aux}) \leftarrow \mathcal{A}^{(\theta_1, \dots, \theta_\nu)}(\text{ck}) \\ \omega \leftarrow \{0, 1\}^{\text{poly}(\lambda)} \\ \text{cm} := \text{CM.Commit}(\text{ck}, m_1; \omega) \end{array} \right. \right\} .$$

Note that in this definition CM does not have access to  $(\theta_1, \dots, \theta_\nu)$ . The above generalizes the notion of a commitment scheme, which is recovered from the above by setting the oracles to be empty.

Moreover, we say that CM is *s-succinct* if for every commitment key  $\text{ck} \in \text{CM.Setup}(1^\lambda)$ , message  $m \in \{0, 1\}^*$ , and randomness  $\omega$ , it holds that  $\text{CM.Commit}(\text{ck}, m; \omega) \in \{0, 1\}^{s(\lambda)}$ .

We have the following simple claim about any binding and hiding commitment scheme.

*Claim.* Let CM be a binding and hiding commitment scheme. Then for every message  $m$ ,

$$\Pr_{\omega, \omega'} \left[ \text{CM.Commit}(\text{ck}, m, \omega) = \text{CM.Commit}(\text{ck}, m, \omega') \right] = \text{negl}(\lambda) .$$

A proof of the above claim appears in Claim 3.4 of [CCS22].

### 3.6 Constraint Detection for Low-Degree Polynomials

**Definition 4.** Let  $\mathbf{d} = (d_1, \dots, d_m) \in \mathbb{N}^m$ . The low-degree polynomial evaluation code is defined as follows:

$$\text{LD}[\mathbb{F}, m, \mathbf{d}] := \left\{ c \in (\mathbb{F}^m \rightarrow \mathbb{F}) : \exists p \in \mathbb{F}^{\leq d}[X_1, \dots, X_m] \text{ s.t. } \forall x \in \mathbb{F}^m, c(x) = p(x) \right\} .$$

Further, let  $\mathcal{F} = \{\mathbb{F}_\lambda\}_{\lambda \in \mathbb{N}}$  be a family of fields,  $m: \mathbb{N} \rightarrow \mathbb{N}$  an arity function, and  $\mathbf{d}: \mathbb{N} \rightarrow \mathbb{N}^m$  a degree function. We define

$$\text{LD}[\mathcal{F}, m, \mathbf{d}] := \left\{ \text{LD}[\mathbb{F}_\lambda, m(\lambda), \mathbf{d}(\lambda)] \right\}_{\lambda \in \mathbb{N}} .$$

We recall the notion of constraints for linear codes.

**Definition 5.** Let  $\mathcal{C} \subseteq (D \rightarrow \mathbb{F})$  be a linear code. A subset  $Q \subseteq D$  is **constrained** if there exists a nonzero  $z: Q \rightarrow \mathbb{F}$  such that, for every  $c \in \mathcal{C}$ ,  $\sum_{x \in Q} z(x)c(x) = 0$  (equivalently, if there exists  $z \neq 0 \in \mathcal{C}^\perp$  with  $\text{supp}(z) \subseteq Q$ ); we refer to  $z$  as a **constraint** on  $Q$ . We say that  $Q$  is **unconstrained** if it is not constrained. We say that  $Q \subseteq D$  **determines**  $x \in D$  if  $x \in Q$  or there exists a constraint  $z$  on  $Q \cup \{x\}$  such that  $z(x) \neq 0$ .

We recall the definition of a constraint detector [BCF+17], which is an algorithm that determines whether a set of queries  $Q$  is constrained and, if so, outputs a constraint.

**Definition 6.** Let  $\mathcal{C} \subseteq (D \rightarrow \mathbb{F})$  be a linear code. An algorithm CD is a **constraint detector** for  $\mathcal{C}$  if, given as input a set  $Q \subseteq D$ , outputs: (i) a basis for the space of constraints  $\{z: Q \rightarrow \mathbb{F} : \forall c \in \mathcal{C}, \sum_{x \in Q} z(x)c(x) = 0\}$  on  $Q$  if  $Q$  is constrained; (ii)  $\perp$  if  $Q$  is unconstrained; A code family  $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$  has **efficient constraint detection** if there exists a polynomial-time algorithm CD such that, for every  $\lambda \in \mathbb{N}$ ,  $\text{CD}(1^\lambda, \cdot)$  is a constraint detector for  $\mathcal{C}_\lambda$ .

The following theorem is proved in [BCF+17]:

**Theorem 1.** *The code family  $\text{LD}[\mathcal{F}, m, \mathbf{d}]$  has a constraint detector  $\text{CD}(1^\lambda, \cdot)$  that runs in time  $\text{poly}(m(\lambda), d(\lambda), \log |\mathbb{F}_\lambda|)$ , where  $d(\lambda) := \max_{i \in [m]} d(\lambda)_i$ . In particular, it has efficient constraint detection.*

### 3.7 Forking Lemmas

We state a general forking lemma proved in [BN06].

**Lemma 1.** *Fix  $t, \lambda \in \mathbb{N}$ . Let  $\mathcal{A}$  be a probabilistic algorithm that on input  $x, y_1, \dots, y_t$  returns a pair  $(I, \sigma)$ , where  $I \in [t]$  and  $\sigma$  is referred to as a side output. Let  $\text{IG}$  be a probabilistic algorithm that we call the input generator. The accepting probability of  $\mathcal{A}$ , denoted  $\text{acc}$ , is defined as follows:*

$$\text{acc} := \Pr \left[ I \geq 1 \mid \begin{array}{l} x \leftarrow \text{IG} \\ y_1, \dots, y_t \leftarrow \mathcal{U}(\{0, 1\}^\lambda) \\ (I, \sigma) \leftarrow \mathcal{A}(x, y_1, \dots, y_t) \end{array} \right].$$

The forking algorithm  $\text{Fork}_{\mathcal{A}}$  associated to  $\mathcal{A}$  is the probabilistic algorithm that takes input  $x$  and proceeds as follows:

- (i) Pick coins  $\rho$  for  $\mathcal{A}$  at random.
- (ii) Sample  $y_1, \dots, y_t \leftarrow \mathcal{U}(\{0, 1\}^\lambda)$ , and run  $\mathcal{A}(x, y_1, \dots, y_t; \rho)$  to obtain  $(I, \sigma)$ .
- (iii) If  $I = 0$  then return  $(0, \varepsilon, \varepsilon)$ .
- (iv) Otherwise, sample  $y'_1, \dots, y'_t \leftarrow \mathcal{U}(\{0, 1\}^\lambda)$  and run  $\mathcal{A}(x, y_1, \dots, y_{I-1}, y'_I, \dots, y'_t; \rho)$  to obtain  $(I', \sigma')$ .
- (v) If  $(I = I' \text{ and } y_I \neq y'_I)$  then return  $(1, \sigma, \sigma')$ .
- (vi) Otherwise return  $(0, \varepsilon, \varepsilon)$ .

Let

$$\text{frk} := \Pr \left[ b = 1 \mid \begin{array}{l} x \leftarrow \text{IG} \\ (b, \sigma, \sigma') \leftarrow \text{Fork}_{\mathcal{A}}(x) \end{array} \right].$$

Then

$$\text{frk} \geq \text{acc} \cdot \left( \frac{\text{acc}}{t} - \frac{1}{2^\lambda} \right),$$

alternatively,

$$\text{acc} \leq \frac{t}{2^\lambda} + \sqrt{t \cdot \text{frk}}.$$

### 3.8 Identical-Until-Bad

We consider two programs,  $G$  and  $H$ , which are written in some pseudocode. We say that  $G$  and  $H$  are *identical-until-bad* if they are syntactically identical except for statements that follow the setting of a bad flag to true. Somewhat more formally, let  $G$  and  $H$  be programs written in some pseudocode and let  $\text{bad}$  be a flag that occurs in both of them. We say that  $G$  and  $H$  are *identical-until-bad* if their code is the same except possibly places where  $G$  has a statement “set the bad flag” followed by some statements  $S_G$

while  $H$  has a corresponding statement “set the bad flag” followed by some statements  $S_H$ , different from  $S_G$ .

We refer the reader to [BR06] for further details and a full formal treatment of the notion of identical-until-bad, which requires specification of the programming language in question to fully formalise. We stress that that identical-until-bad is a purely syntactic requirement.

We state the fundamental lemma of game-playing, which is proved in [BR06].

**Lemma 2.** *Let  $G$  and  $H$  be identical-until-bad programs and let  $\mathcal{A}$  be an adversary. Then*

$$|\Pr[\mathcal{A}^G = 1] - \Pr[\mathcal{A}^H = 1]| \leq \Pr[\mathcal{A}^G \text{ sets bad}] .$$

## 4 Arithmetized Random Oracle Model

We define the arithmetized random oracle model. As a first step, we define the arithmetized random oracle *distribution*, which is defined over tuples  $(\text{ro}, \text{wo}, \widehat{\text{vo}})$ , and explain how the oracles  $(\text{ro}, \text{wo}, \widehat{\text{vo}})$  are sampled.

**Definition 7.** *Let  $m \in \mathbb{N}$  be an arity parameter,  $\lambda \in \mathbb{N}$  be a security parameter,  $r \in \mathbb{N}$  be a randomness-size parameter,  $w \in \mathbb{N}$  be a witness-size parameter, and  $d \in \mathbb{N}$  be a degree parameter. For all oracle circuits  $B: \{0, 1\}^{m+r} \rightarrow \{0, 1\}^w$ , we define an **arithmetized random oracle distribution**  $\text{ARO}[\mathbb{F}, m, \lambda, d, B]$ ,<sup>10</sup> where  $\mathbb{F}$  is a finite field and the support of  $\text{ARO}[\mathbb{F}, m, \lambda, d, B]$  contains triples  $(\text{ro}, \text{wo}, \widehat{\text{vo}})$  that are sampled as follows:*

1. Sample the **random oracle**  $\text{ro}$  uniformly at random from  $(\{0, 1\}^m \rightarrow \{0, 1\}^\lambda)$ .
2. For every  $x \in \{0, 1\}^m$ , sample a random string  $\mu_x \in \{0, 1\}^r$ . Then define the **witness oracle**  $\text{wo}: \{0, 1\}^m \rightarrow \{0, 1\}^w$  as  $\text{wo}(x) := B^{\text{ro}}(x, \mu_x)$ .
3. Define the **verification function**  $\text{vo}: \{0, 1\}^{m+\lambda+w} \rightarrow \{0, 1\}$  as

$$\text{vo}(x, y, z) := \begin{cases} 1 & \text{if } \text{ro}(x) = y \wedge \text{wo}(x) = z \\ 0 & \text{o.w.} \end{cases} .$$

4. Sample the **(extended) verification oracle**  $\widehat{\text{vo}}: \mathbb{F}^{m+\lambda+w} \rightarrow \mathbb{F}$  uniformly at random from the set

$$\{p \in \mathbb{F}^{\leq d}[X_1, \dots, X_{m+\lambda+w}] : p \text{ equals } \text{vo} \text{ on } \{0, 1\}^{m+\lambda+w}\} .$$

5. Output  $(\text{ro}, \text{wo}, \widehat{\text{vo}})$ .

Next, we define a *family* of ARO distributions, which is parameterized by a family of finite fields  $\mathcal{F} = \{\mathbb{F}_\lambda\}_{\lambda \in \mathbb{N}}$  and a family of oracle circuits  $\mathcal{B} = \{B_\lambda^{(\cdot)}: \{0, 1\}^{m(\lambda)} \rightarrow \{0, 1\}^{w(\lambda)}\}_{\lambda \in \mathbb{N}}$ . Here,  $\mathcal{B}$  can be interpreted as the set of all possible adversarial strategies for learning information about the random oracle, and  $\lambda$  is the security parameter.

<sup>10</sup> Given  $m \in \mathbb{N}$  and the oracle circuit  $B$ , the randomness length  $r$  and witness size  $w$  parameters are determined. Thus  $r, w$  do not appear in the parameterization of ARO.

**Definition 8.** Let  $\mathcal{F} = \{\mathbb{F}_\lambda\}_{\lambda \in \mathbb{N}}$  be a family of fields,  $m: \mathbb{N} \rightarrow \mathbb{N}$  be an arity function,  $w: \mathbb{N} \rightarrow \mathbb{N}$  be a witness-size function,  $\mathcal{B} = \{B_\lambda^{(\cdot)}: \{0, 1\}^{m(\lambda)} \rightarrow \{0, 1\}^{w(\lambda)}\}_{\lambda \in \mathbb{N}}$  be a family of oracle circuits, and  $d: \mathbb{N} \rightarrow \mathbb{N}$  be a degree function. We define the **arithmetized random oracle family** as

$$\text{ARO}[\mathcal{F}, m, d, \mathcal{B}] := \{ \text{ARO}[\mathbb{F}_\lambda, m(\lambda), \lambda, d(\lambda), B_\lambda^{(\cdot)}] \}_{\lambda \in \mathbb{N}} .$$

The “arithmetized random oracle” is the set of all ARO distributions for polynomial-sized circuit families  $\mathcal{B}$ .

**Definition 9.** Let  $\mathcal{F} = \{\mathbb{F}_\lambda\}_{\lambda \in \mathbb{N}}$  be a family of fields,  $m: \mathbb{N} \rightarrow \mathbb{N}$  be an arity function,  $w: \mathbb{N} \rightarrow \mathbb{N}$  be a witness-size function, and  $d: \mathbb{N} \rightarrow \mathbb{N}$  be a degree function. Then, we define a **set of arithmetized random oracle families** as

$$\text{ARO}[\mathcal{F}, m, d] := \{ \text{ARO}[\mathcal{F}, m, d, \mathcal{B}] : \mathcal{B} \text{ is a family of } \text{poly}(\lambda) \text{-size oracle circuits} \},$$

where above  $\mathcal{B} = \{B_\lambda^{(\cdot)}: \{0, 1\}^{m(\lambda)} \rightarrow \{0, 1\}^{w(\lambda)}\}_{\lambda \in \mathbb{N}}$ .

**Acknowledgments.** We thank Giacomo Fenzi for pointing out some inaccuracies in an earlier draft of this paper.

Tom Gur is supported by the UKRI Future Leaders Fellowship MR/S031545/1 and EPSRC New Horizons Grant EP/X018180/1. Jack O’Connor is supported by the Engineering and Physical Sciences Research Council through the Mathematics of Systems Centre for Doctoral Training at the University of Warwick (reference EP/S022244/1). Megan Chen is supported by DARPA under Agreement No. HR00112020023.

## References

- [Alo99] Alon, N.: Combinatorial nullstellensatz. In: Combinatorics, Probability and Computing (1999)
- [AW09] Aaronson, S., Wigderson, A.: Algebrization: a new barrier in complexity theory. *ACM Trans. Comput. Theory* **1**(1), 1–54 (2009)
- [BCCT13] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKs and proof-carrying data. In: *STOC 2013* (2013)
- [BCF+17] Ben-Sasson, E., Chiesa, A., Forbes, M.A., Gabizon, A., Riabzev, M., Spooner, N.: Zero knowledge protocols from succinct constraint detection. In: Kalai, Y., Reyzin, L. (eds.) *TCC 2017*. LNCS, vol. 10678, pp. 172–206. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-70503-3\\_6](https://doi.org/10.1007/978-3-319-70503-3_6)
- [BCL+21] Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. In: Malkin, T., Peikert, C. (eds.) *CRYPTO 2021*. LNCS, vol. 12825, pp. 681–710. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84242-0\\_24](https://doi.org/10.1007/978-3-030-84242-0_24)
- [BCMS20] Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Proof-carrying data from accumulation schemes. In: *TCC 2020* (2020)
- [BCTV14] 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](https://doi.org/10.1007/978-3-662-44381-1_16)











- [BDFG21] Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo Infinite: proof-carrying data from additive polynomial commitments. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 649–680. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84242-0\\_23](https://doi.org/10.1007/978-3-030-84242-0_23)
- [BGH19] Bowe, S., Grigg, J., Hopwood, D.: Halo: recursive proof composition without a trusted setup. ePrint Report 2019/1021 (2019)
- [BGV11] Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable delegation of computation over large datasets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 111–131. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22792-9\\_7](https://doi.org/10.1007/978-3-642-22792-9_7)
- [BMRS20] Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Coda: decentralized cryptocurrency at scale. ePrint Report 2020/352 (2020)
- [BN06] Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: CCS 2006 (2006)
- [BR06] Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006). [https://doi.org/10.1007/11761679\\_25](https://doi.org/10.1007/11761679_25)
- [BR93] Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: CCS 1993 (1993)
- [CCDW20] Chen, W., Chiesa, A., Dauterman, E., Ward, N.P.: Reducing participation costs via incremental verification for ledger systems. Cryptology ePrint Archive, Report 2020/1522 (2020)
- [CCS22] Chen, M., Chiesa, A., Spooner, N.: On succinct non-interactive arguments in relativized worlds. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022. LNCS, vol. 13276, pp. 336–366. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-07085-3\\_12](https://doi.org/10.1007/978-3-031-07085-3_12)
- [CFS17] Chiesa, A., Forbes, M.A., Spooner, N.: A zero knowledge sumcheck and its applications. ePrint Report 2017/305 (2017)
- [CL20] Chiesa, A., Liu, S.: On the impossibility of probabilistic proofs in relativized worlds. In: ITCS 2020 (2020)
- [COS20] Chiesa, A., Ojha, D., Spooner, N.: FRACTAL: post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 769–793. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45721-1\\_27](https://doi.org/10.1007/978-3-030-45721-1_27)
- [CT10] Chiesa, A., Tromer, E.: Proof-carrying data and hearsay arguments from signature cards. In: ICS 2010 (2010)
- [CTV13] Chong, S., Tromer, E., Vaughan, J.A.: Enforcing language semantics using proof-carrying data. ePrint Report 2013/513 (2013)
- [CTV15] Chiesa, A., Tromer, E., Virza, M.: Cluster computing in zero knowledge. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 371–403. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46803-6\\_13](https://doi.org/10.1007/978-3-662-46803-6_13)
- [GK03] Goldwasser, S., Kalai, Y.T.: On the (in)security of the Fiat-Shamir paradigm. In: FOCS 2003 (2003)
- [Gro16] 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](https://doi.org/10.1007/978-3-662-49896-5_11)
- [GW11] Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: STOC 2011 (2011)
- [HN22] Hall-Andersen, M., Nielsen, J.B.: On valiant’s conjecture: impossibility of incrementally verifiable computation from random oracles. Cryptology ePrint Archive, Paper 2022/542 (2022)

- [JKRS09] Juma, A., Kabanets, V., Rackoff, C., Shpilka, A.: The black-box query complexity of polynomial summation. *Comput. Complex.* **18**, 59–79 (2009). <https://doi.org/10.1007/s00037-009-0263-7>
- [JLLW22] Jain, A., Lin, H., Luo, J., Wichs, D.: The pseudorandom oracle model and ideal obfuscation. *Cryptology ePrint Archive*, Paper 2022/1204 (2022)
- [KB20] Kattis, A., Bonneau, J.: Proof of necessary work: succinct state verification with fairness guarantees. *ePrint Report 2020/190* (2020)
- [KR08] Kalai, Y.T., Raz, R.: Interactive PCP. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008*. LNCS, vol. 5126, pp. 536–547. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-70583-3\\_44](https://doi.org/10.1007/978-3-540-70583-3_44)
- [KST22] Kothapalli, A., Setty, S., Tzialla, I.: Nova: recursive zero-knowledge arguments from folding schemes. In: Dodis, Y., Shrimpton, T. (eds.) *CRYPTO 2022*. LNCS, vol. 13510, pp. 359–388. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-15985-5\\_13](https://doi.org/10.1007/978-3-031-15985-5_13)
- [Mic00] Micali, S.: Computationally sound proofs. *SIAM J. Comput.* **30**(4), 1253–1298 (2000). Preliminary version appeared in FOCS 1994
- [Mina] O(1) Labs: Mina Cryptocurrency. <https://minaprotocol.com/>
- [NT16] Naveh, A., Tromer, E.: PhotoProof: cryptographic image authentication for any set of permissible transformations. In: *S&P 2016* (2016)
- [TFZ+22] Tyagi, N., Fisch, B., Zitek, A., Bonneau, J., Tessaro, S.: VerRSA: verifiable registries with efficient client audits from RSA authenticated dictionaries. In: *CCS 2022* (2022)
- [Val08] Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) *TCC 2008*. LNCS, vol. 4948, pp. 1–18. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78524-8\\_1](https://doi.org/10.1007/978-3-540-78524-8_1)
- [Zha22] Zhandry, M.: Augmented random oracles. In: Dodis, Y., Shrimpton, T. (eds.) *CRYPTO 2022*. LNCS, vol. 13509, pp. 35–65. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-15982-4\\_2](https://doi.org/10.1007/978-3-031-15982-4_2)



# Supersingular Curves You Can Trust

Andrea Basso<sup>1,2</sup> , Giulio Codogni<sup>3</sup> , Deirdre Connolly<sup>4</sup>, Luca De Feo<sup>5</sup> ,  
Tako Boris Fouotsa<sup>6</sup> , Guido Maria Lido<sup>3</sup> , Travis Morrison<sup>7</sup> ,  
Lorenz Panny<sup>8</sup>, Sikhhar Patranabis<sup>9</sup> , and Benjamin Wesolowski<sup>10,11,12</sup> 

<sup>1</sup> University of Birmingham, Birmingham, UK  
`andrea.basso@bristol.ac.uk`

<sup>2</sup> University of Bristol, Bristol, UK

<sup>3</sup> Dipartimento di Matematica, Università degli Studi di Roma Tor Vergata, Via della Ricerca Scientifica, 00133 Roma, Italy  
`codogni@mat.uniroma2.it`, `guidomaria.lido@uniroma2.it`

<sup>4</sup> Zcash Foundation, Mclean, USA

<sup>5</sup> IBM Research Europe, Zürich, Switzerland  
`eurocrypt23@defeo.lu`

<sup>6</sup> EPFL, Lausanne, Switzerland  
`tako.fouotsa@epfl.ch`

<sup>7</sup> Virginia Tech, Blacksburg, VA, USA  
`tmo@vt.edu`

<sup>8</sup> Academia Sinica, Taipei, Taiwan  
`lorenz@yx7.cc`

<sup>9</sup> IBM Research India, Bangalore, India  
`sikhhar.patranabis@ibm.com`

<sup>10</sup> Univ. Bordeaux, CNRS, Bordeaux INP, IMB, UMR 5251, 33400 Talence, France  
`benjamin.wesolowski@math.u-bordeaux.fr`

<sup>11</sup> INRIA, IMB, UMR 5251, 33400 Talence, France

<sup>12</sup> ENS de Lyon, CNRS, UMPA, UMR 5669, Lyon, France

**Abstract.** Generating a supersingular elliptic curve such that nobody knows its endomorphism ring is a notoriously hard task, despite several isogeny-based protocols relying on such an object. A trusted setup is often proposed as a workaround, but several aspects remain unclear. In this work, we develop the tools necessary to practically run such a distributed trusted-setup ceremony.

Our key contribution is the first statistically zero-knowledge proof of isogeny knowledge that is compatible with any base field. To prove

---

Author list in alphabetical order; see <https://ams.org/profession/leaders/CultureStatement04.pdf>. This work began at the Banff International Research Station workshop “Supersingular Isogeny Graphs in Cryptography” (21w5229). This research was funded in part by the MIUR Excellence Department Project MatMod@TOV awarded to the Department of Mathematics, University of Rome Tor Vergata, the Commonwealth Cyber Initiative, the Academia Sinica Investigator Award AS-IA-109-M01, the Agence Nationale de la Recherche under grant ANR MELODIA (ANR-20-CE40-0013), and the France 2030 program under grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

© International Association for Cryptologic Research 2023

C. Hazay and M. Stam (Eds.): EUROCRYPT 2023, LNCS 14005, pp. 405–437, 2023.

[https://doi.org/10.1007/978-3-031-30617-4\\_14](https://doi.org/10.1007/978-3-031-30617-4_14)

statistical ZK, we introduce isogeny graphs with Borel level structure and prove they have the Ramanujan property. Then, we analyze the security of a distributed trusted-setup protocol based on our ZK proof in the simplified universal composability framework. Lastly, we develop an optimized implementation of the ZK proof, and we propose a strategy to concretely deploy the trusted-setup protocol.

**Keywords:** Isogenies · Ramanujan Graphs · Zero-knowledge Proofs · Trusted Setup

## 1 Introduction

Be it foundationally or for efficiency, most of isogeny-based cryptography is built upon supersingular elliptic curves [15, 17, 22, 27, 28, 37, 42]. At the heart of it, lies the *supersingular isogeny graph*: a graph whose vertices represent supersingular elliptic curves (up to isomorphism) and whose edges represent isogenies (up to isomorphism) of some fixed small prime degree between them. A foundational hard problem for isogeny-based cryptography is then: given two supersingular elliptic curves, find a path in the supersingular isogeny graph connecting them.

An endomorphism is an isogeny from a curve  $E$  to itself, and their collection forms the *endomorphism ring*  $\text{End}(E)$ . In recent years, the connection between finding isogeny paths and computing endomorphism rings of supersingular curves has become increasingly important [32, 35, 58, 59]. It is now established that, assuming the generalized Riemann hypothesis, there exists probabilistic polynomial time algorithms for these two problems:

1. Given supersingular elliptic curves  $E_0, E_1$  along with descriptions of their endomorphism rings, compute an isogeny path  $E_0 \rightarrow E_1$ ;
2. Given a supersingular elliptic curve  $E_0$  along with a description of its endomorphism ring, and given an isogeny path  $E_0 \rightarrow E_1$ , compute a description of the endomorphism ring of  $E_1$ .

These algorithms—and variants—have successfully been used both constructively [22, 27, 37] and for cryptanalysis [28, 32, 34, 35, 49, 51].

Without the additional information above, computing the endomorphism ring of an arbitrary supersingular curve remains a hard problem, both for classical and quantum computers. Given the importance of this problem, it is natural to ask whether it is possible to sample supersingular curves such that computing their endomorphism ring is a hard problem, crucially, even for the party who does the sampling. We shall call these objects *Supersingular Elliptic Curves of Unknown Endomorphism Ring*, or **SECUER**<sup>1</sup> in short.

**Applications.** Generating a SECUER has turned out to be a delicate task, and no such curve has ever been generated. Yet, several isogeny-based schemes can

---

<sup>1</sup> The British spelling is **SECURE**.

only be instantiated with a SECUER. This is the case, for example, of isogeny-based verifiable delay functions [28] and delay encryption [12]. The so-called CGL hash function based on supersingular curves [17] has been shown to be broken by the knowledge of the endomorphism ring [32], and one possible fix is to instantiate it with a SECUER. Other protocols which require a SECUER include hash proof systems, dual mode PKE [1], oblivious transfer [44], OPRF [4], and commitment schemes [54].

**Contributions.** We analyze and put into practice a protocol for distributed generation of SECUERS. Our main technical contribution is a key ingredient of the protocol: a new proof of isogeny knowledge (two curves  $E_0$  and  $E_1$  being public, a party wishes to prove that they know an isogeny  $E_0 \rightarrow E_1$  without revealing it). Our proof is similar to the SIDH proof of knowledge [23, 25], but extends it in a way that makes it compatible with any base field, any walk length, and has provable statistical zero-knowledge (unlike any previous proof of isogeny knowledge). In particular, its statistical security makes it fully immune to the recent attacks [14, 46, 52].

To prove statistical security, we analyze *supersingular  $\ell$ -isogeny graphs with level structure*, a generalization of isogeny graphs that was recently considered in [3, 27]. We prove that these graphs, like classic isogeny graphs, possesses the Ramanujan property, a fact that is of independent interest. Using the property, we analyze the mixing behavior of random walks, which lets us give very precise parameters to instantiate the proof of knowledge at any given security level.

To show that the resulting protocol is practical, we implement it on top of Microsoft's SIDH library<sup>2</sup> and benchmark it for each of the standard SIKE primes [41]. We must stress that SIDH-style primes are possibly the most favorable to our protocol, in terms of practical efficiency.

Finally, we sketch a roadmap to run the distributed generation protocol for the SIKE primes in a real world setting with hundreds of participants.

*Limitations.* We must point out that our new proof of knowledge is not well adapted to a secure distributed generation protocol in the case where one wants to generate a SECUER defined over a prime field  $\mathbb{F}_p$ , instead of  $\mathbb{F}_{p^2}$ , such as in [1, 44]. Different proofs of knowledge [8, 24] could be plugged in the distributed protocol for the  $\mathbb{F}_p$  case, however their practical usability is dubious.

## 1.1 Generating a SECUER

The cornerstone of isogeny-based cryptography is the endomorphism ring problem: if it could be solved efficiently, then all of supersingular isogeny-based cryptography would be broken [32, 35, 58], leaving only ordinary isogeny-based cryptography [21, 26, 55] standing.

**Definition 1 (Endomorphism ring problem).** *Given a supersingular curve  $E/\mathbb{F}_{p^2}$ , compute its endomorphism ring  $\text{End}(E)$ . That is, compute an integral*

<sup>2</sup> <https://github.com/microsoft/PQCrypto-SIDH>.

basis for a maximal order  $\mathcal{O}$  of the quaternion algebra ramified at  $p$  and  $\infty$ , as well as an explicit isomorphism  $\mathcal{O} \simeq \text{End}(E)$ .

For any  $p$ , there exists a polynomially sized subset of all supersingular curves for which the endomorphism ring can be computed in polynomial time [16, 45], but the problem is believed to be exponentially hard in general, even for quantum computers. A related problem, commonly encountered in isogeny protocols, is finding paths in supersingular isogeny graphs.

**Definition 2 (Isogeny  $\ell$ -walk problem).** *Given two supersingular curves  $E, E'/\mathbb{F}_{p^2}$  of the same order, and a small prime  $\ell$ , find a walk from  $E$  to  $E'$  in the  $\ell$ -isogeny graph.*

Such walks are always guaranteed to exist, as soon as they have length in  $O(\log(p))$  [17, 43, 47, 50].

The two problems are known to be polynomial time equivalent, assuming GRH [59]. Indeed, given  $\text{End}(E)$  and  $\text{End}(E')$ , it is easy to compute a path  $E \rightarrow E'$ . Reciprocally, given  $\text{End}(E)$  and a path  $E \rightarrow E'$ , it is easy to compute  $\text{End}(E')$ ; and, by random self-reducibility, we can always assume that one of  $\text{End}(E)$  or  $\text{End}(E')$  is known.

Our goal is to generate a SECUER: a curve for which the endomorphism ring problem is hard, and consequently one for which it is hard to find a path to any other given curve.

**What Does Not Work.** The supersingular elliptic curves over a finite field  $k$  of characteristic  $p$  are those such that  $\#E(k) = 1 \pmod p$ . Any supersingular curve is isomorphic to one defined over a field with  $p^2$  elements, thus, without loss of generality, we are only interested in supersingular curves defined over  $\mathbb{F}_{p^2}$ . Among the  $p^2$  isomorphism classes of elliptic curves over  $\mathbb{F}_{p^2}$ , only  $\approx p/12$  of them correspond to supersingular curves.

The standard way to construct supersingular curves is to start from a curve with complex multiplication over a number field, and then reduce modulo  $p$ . Complex multiplication elliptic curves have supersingular reduction modulo 50% of the primes, thus this technique quickly produces supersingular curves for almost all primes. For example, the curve  $y^2 = x^3 + x$ , which has complex multiplication by the ring  $\mathbb{Z}[i]$  of Gaussian integers, is supersingular modulo  $p$  if and only if  $p = 3 \pmod 4$ . Most isogeny-based protocols are instantiated using precisely this curve as starting point. These curves are not SECUERS, though, because from the information on complex multiplication one can compute the endomorphism ring in polynomial time [16, 45].

As  $p$  grows, the curves with computable<sup>3</sup> complex multiplication form only a negligible fraction of all supersingular curves in characteristic  $p$ , so we may still hope to get a SECUER if we can sample a supersingular curve at random from the

---

<sup>3</sup> Deuring showed that any supersingular curve can be lifted in several ways to a curve with complex multiplication, but for almost all curves computing such lifts has complexity exponential in  $\log(p)$ .

whole set. The natural way to do so is to start from a well known supersingular curve, e.g.  $E_0 : y^2 = x^3 + x$ , take a random walk  $E_0 \rightarrow E_1$  in the isogeny graph, and then select the arrival curve  $E_1$ . But, by virtue of the reductions mentioned above, any  $E_1$  constructed this way cannot be called a SECUER either.

Several other techniques have been considered for generating SECUERS, however all attempts have failed so far [10, 48].

**Distributed Generation of SECUERS.** An obvious solution that has been proposed for schemes that need a SECUER is to rely on a trusted party to start from a special curve  $E_0$  and to perform an isogeny walk to a random curve  $E_1$ . Although  $E_1$  is not a SECUER, if the trusted party keeps the walk  $E_0 \rightarrow E_1$  secret, no one else will be able to compute  $\text{End}(E_1)$ .

Of course, relying on a trusted third party is undesirable. The natural next step is to turn this idea into a distributed protocol with  $t$  parties generating a sequence of walks  $E_0 \rightarrow E_1 \rightarrow E_2 \rightarrow \dots \rightarrow E_t$ . First, suppose that the sequence was generated honestly: the  $i$ -th party indeed generated a random isogeny from the previous curve  $E_{i-1}$  to a new curve  $E_i$ . Then it is sufficient for a *single* party to honestly discard their isogeny, for no path to be known by *anyone* from  $E_0$  to  $E_t$ . Then,  $E_t$  is a SECUER for all practical purposes.

To make this protocol secure against active adversaries, an additional ingredient is needed. As it is, the last party could cheat as follows: instead of generating an isogeny  $E_{t-1} \rightarrow E_t$ , they could reboot the chain by generating an isogeny  $E_0 \rightarrow E_t$ , and submitting that instead. They could then compute the endomorphism ring of  $E_t$ . If only the curves  $E_i$  along the path are revealed, it is impossible to detect such misbehavior. To prevent this, each party needs to prove that they know their component of the walk: an isogeny  $E_{i-1} \rightarrow E_i$  (as first discussed in [12]). To this end, we develop a statistically zero-knowledge proof of isogeny knowledge.

## 1.2 Proof of Isogeny Knowledge

**State-of-the-Art.** Protocols to prove knowledge of an isogeny have been mostly studied for signatures. The first such protocol is the SIDH-based proof of knowledge of [25]. Its security proof was found to be flawed and then fixed, either by changing the assumptions [38] or by changing the protocol [23]. However, these protocols are now fully broken by the recent polynomial time attacks on SIDH-like protocols [14, 46, 52]. These attacks can be avoided by relying on ternary challenges [9, 23].

CSIDH-based proofs of knowledge were first introduced in [24], and then improved in [8] for the parameter set CSIDH-512. These are limited to isogeny walks between curves defined over a prime field  $\mathbb{F}_p$ , and tend to be prohibitively slow outside of the specially prepared parameter set CSIDH-512.

Finally, De Feo and Burdges propose an efficient proof of knowledge tailored to finite fields used in delay protocols [12]. However the soundness of this protocol is only conjectural, and, being based on pairing assumptions, is broken by quantum computers.

In summary, no general purpose, quantum-safe, zero-knowledge proof of knowledge of an isogeny walk between supersingular curves defined over  $\mathbb{F}_{p^2}$  exists in previous literature.

**Overview of Our Method.** Our main technical contribution is a new proof of knowledge that ticks all the boxes above: it is compatible with any base field, any walk length, it has provable statistical zero-knowledge, and is practical—as illustrated by our implementation. The idea is the following. Two elliptic curves  $E_0$  and  $E_1$  being public, some party, the prover, wishes to convince the verifier that they know an isogeny  $\phi : E_0 \rightarrow E_1$  (of degree, say,  $2^m$ , large enough so it is guaranteed that such an isogeny exists). First, the prover secretly generates a random isogeny walk  $\psi : E_0 \rightarrow E_2$  of degree, say,  $3^n$ . Defining  $\phi'$  with kernel  $\psi(\ker(\phi))$ , and  $\psi'$  with kernel  $\phi(\ker(\psi))$ , one obtains the following commutative diagram, known as “SIDH square” in the literature:

$$\begin{array}{ccc}
 E_0 & \xrightarrow{\phi} & E_1 \\
 \psi \downarrow & & \downarrow \psi' \\
 E_2 & \xrightarrow{\phi'} & E_3
 \end{array} \tag{1}$$

Now, the prover publishes a hiding and binding commitment to  $E_2$  and  $E_3$ . The verifier may now ask the prover to reveal one of the three isogenies  $\psi$ ,  $\phi'$ , or  $\psi'$ , by drawing a random  $\text{chall} \in \{-1, 0, 1\}$  (and open the commitment(s) corresponding to the relevant endpoints). For the prover to succeed with overwhelming probability, they must know all three answers, so they must know an isogeny from  $E_0$  to  $E_1$ : the composition  $\psi' \circ \phi' \circ \psi : E_0 \rightarrow E_1$ . This is the idea behind the soundness of the protocol.

So far, this protocol is more or less folklore and superficially similar to [23, §5.3]. But does it leak any information? Whereas previous protocols only achieved computational zero-knowledge, we provide a tweak that achieves statistical zero-knowledge: there is a simulator producing transcripts that are statistically indistinguishable from a valid run of the protocol. The simulator starts by choosing the challenge  $\text{chall}$  *first*, then it generates an isogeny that is statistically indistinguishable from either  $\psi$ ,  $\phi'$ , or  $\psi'$ , according to the value of  $\text{chall}$ . Simulating  $\psi$  (or  $\psi'$ ) is straightforward: generate a random isogeny walk  $\tilde{\psi}$  (or  $\tilde{\psi}'$ ) of degree  $3^n$  from  $E_0$  (or from  $E_1$ ). The isogeny  $\tilde{\psi}$  is a *perfect* simulation of  $\psi$ . Simulating  $\phi'$  seems trickier. An obvious approach is to first generate a random  $E_2$  (for instance, by simulating  $\psi : E_0 \rightarrow E_2$ ), then generate a random walk isogeny  $\tilde{\phi}' : E_2 \rightarrow E_3$  of degree  $2^m$ . While this may seem too naive, we in fact prove that when  $\deg(\psi)$  is large enough, the distribution of  $\tilde{\phi}'$  is statistically close to a honestly generated  $\phi'$ . The key is a proof that the isogeny graph enriched with so-called *level structure* has rapid mixing properties.

**Isogeny Graphs with Level Structure.** The isogeny  $\phi'$  is essentially characterized by its source,  $E_2$ , and its kernel  $\ker(\phi')$ , a (cyclic) subgroup of order



$\deg(\phi')$ . We are thus interested in random variables of the form  $(E, C)$ , where  $E$  is an elliptic curve, and  $C$  a cyclic subgroup of  $E$ , of order some integer  $d$  (not divisible by  $p$ ). We call such a pair  $(E, C)$  a *level  $d$  Borel structure*.

The simulator proposed above essentially generates  $\tilde{\phi}'$  as a uniformly random level  $2^m$  Borel structure  $(E, C) = (E_2, \ker(\tilde{\phi}'))$ . On the other hand, a honestly generated  $\phi'$  corresponds to a pair  $(\psi(E_0), \psi(\ker \phi))$ , and  $\psi$  is a uniformly random isogeny walk of degree  $3^n$ . This process corresponds to a random walk of length  $n$  in the *3-isogeny graph with level  $2^m$  structure*, with starting point  $(E_0, \ker \phi)$ . We prove the following result.

**Theorem 3.** *Let  $G = G(p, d, \ell)$  the supersingular  $\ell$ -isogeny graph with level  $d$  Borel structure. The adjacency matrix  $A$  of  $G$  is diagonalizable, with real eigenvalues, and has the Ramanujan property, i.e. the integer  $\ell + 1$  is an eigenvalue of  $A$  of multiplicity one, while all the other eigenvalues are contained in the Hasse interval  $[-2\sqrt{\ell}, 2\sqrt{\ell}]$ .*

As a consequence, we prove that random walks quickly converge to the stationary distribution, so  $\tilde{\phi}'$  and  $\phi'$  are statistically indistinguishable.

*Paper outline.* We start in Sect. 2 with a few technical preliminaries on elliptic curves, isogenies, and proofs of knowledge. Section 3 is dedicated to the proof of Theorem 3. This section can be read independently from the rest. The reader only interested in applications, and willing to accept Theorem 3 (and its consequence on non-backtracking random walks, Theorem 11, page 14), can safely skip to the following section. This theoretical tool at hand, we then describe and analyse the new proof of isogeny knowledge in Sect. 4. We describe the protocol to generate a SECUER in Sect. 5, and prove its security. Finally, we report on our implementation in Sect. 6.

## 2 Preliminaries

### 2.1 General Notations

We write  $x \leftarrow \mathcal{X}$  to represent that an element  $x$  is sampled at random from a set/distribution  $\mathcal{X}$ . The output  $x$  of a deterministic algorithm  $\mathcal{A}$  is denoted by  $x = \mathcal{A}$  and the output  $x'$  of a randomized algorithm  $\mathcal{A}'$  is denoted by  $x' \leftarrow \mathcal{A}'$ . For  $a, b \in \mathbb{N}$  such that  $a, b \geq 1$ , we denote by  $[a, b]$  (resp.  $[a]$ ) the set of integers lying between  $a$  and  $b$ , both inclusive (the set of integers lying between 1 and  $a$ , both inclusive). We refer to  $\lambda \in \mathbb{N}$  as the security parameter, and denote by  $\text{poly}(\lambda)$ ,  $\text{polylog}(\lambda)$  and  $\text{negl}(\lambda)$  any generic (unspecified) polynomial, poly-logarithmic or negligible function in  $\lambda$ , respectively.<sup>4</sup> For probability distributions  $\mathcal{X}$  and  $\mathcal{Y}$ , we write  $\mathcal{X} \approx \mathcal{Y}$  if the statistical distance between  $\mathcal{X}$  and  $\mathcal{Y}$  is negligible.

<sup>4</sup> A function  $f : \mathbb{N} \rightarrow \mathbb{N}$  is said to be negligible in  $\lambda$  if for every positive polynomial  $p$ ,  $f(\lambda) < 1/p(\lambda)$  when  $\lambda$  is sufficiently large.

## 2.2 Elliptic Curves, Isogenies and “SIDH Squares”

We assume the reader has some familiarity with elliptic curves and isogenies. Throughout the text,  $p$  shall be a prime number,  $\mathbb{F}_p$  and  $\mathbb{F}_{p^2}$  the finite fields with  $p$  and  $p^2$  elements respectively. Unless specified otherwise, all elliptic curves will be supersingular and defined over  $\mathbb{F}_{p^2}$ . We write  $E[d]$  for the subgroup of  $d$ -torsion points of  $E$  over the algebraic closure.

Unless specified otherwise, all isogenies shall be separable. If  $G$  is a finite subgroup of  $E$ , we write  $\phi : E \rightarrow E/G$  for the unique (up to post-composition with an isomorphism of  $E/G$ ) separable isogeny with kernel  $G$ . If  $G$  is cyclic, we say the isogeny is cyclic. We denote by  $\hat{\phi}$  the dual isogeny to  $\phi$ . Separable isogenies and their duals can be computed and/or evaluated in time  $\text{poly}(\#G)$  using any of the algorithms in [7, 56], however in some cases, e.g. when  $\#G$  only contains small factors, this cost may be lowered to as little as  $\text{polylog}(\#G)$ .

Given separable isogenies  $\phi : E_0 \rightarrow E_1$  and  $\psi : E_0 \rightarrow E_2$  of coprime degrees, we obtain the commutative diagram in (1) by defining  $\phi' : E_2 \rightarrow E_2/\psi(\ker(\phi))$  and  $\psi' : E_1 \rightarrow E_1/\phi(\ker(\psi))$ . Again,  $E_3$  is only defined up to isomorphism. In categorical parlance, this is the *pushout* of  $\phi$  and  $\psi$ , but cryptographers may know it better through its use in the SIDH key exchange. We refer to these commutative diagrams as *SIDH squares* or *SIDH ladders* (see Sect. 4.2 for more details).

## 2.3 Proofs of Knowledge

Our main technical contribution is a  $\Sigma$ -protocol to prove knowledge of an isogeny of given degree between two supersingular elliptic curves. Recall a  $\Sigma$ -protocol for an NP-language  $\mathcal{L}$  is a public-coin three-move interactive proof system consisting of two parties: a verifier and a prover. The prover is given a witness  $w$  for an element  $x \in \mathcal{L}$ , his goal is to convince the verifier that he knows  $w$ .

**Definition 4 ( $\Sigma$ -protocol).** A  $\Sigma$ -protocol  $\Pi_\Sigma$  for a family of relations  $\{\mathcal{R}\}_\lambda$  parameterized by security parameter  $\lambda$  consists of PPT algorithms  $(P_1, P_2, V)$  where  $V$  is deterministic and we assume  $P_1, P_2$  share states. The protocol proceeds as follows:

1. The prover, on input  $(x, w) \in \mathcal{R}$ , returns a commitment  $\text{com} \leftarrow P_1(x, w)$  which is sent to the verifier.
2. The verifier flips  $\lambda$  coins and sends the result to the prover.
3. Call  $\text{chall}$  the message received from the verifier, the prover runs  $\text{resp} \leftarrow P_2(\text{chall})$  and returns  $\text{resp}$  to the verifier.
4. The verifier runs  $V(x, \text{com}, \text{chall}, \text{resp})$  and outputs a bit.

A transcript  $(\text{com}, \text{chall}, \text{resp})$  is said to be valid, or accepting, if  $V(x, \text{com}, \text{chall}, \text{resp})$  outputs 1. The main requirements of a  $\Sigma$ -protocol are:

**Correctness:** If the prover knows  $(x, w) \in \mathcal{R}$  and behaves honestly, then the verifier outputs 1.

**$n$ -special soundness:** There exists a polynomial-time extraction algorithm that, given a statement  $x$  and  $n$  valid transcripts

$$(\text{com}, \text{chall}_1, \text{resp}_1), \dots, (\text{com}, \text{chall}_n, \text{resp}_n)$$

where  $\text{chall}_i \neq \text{chall}_j$  for all  $1 \leq i < j \leq n$ , outputs a witness  $w$  such that  $(x, w) \in \mathcal{R}$  with probability at least  $1 - \varepsilon$  for soundness error  $\varepsilon$ .

A special sound  $\Sigma$ -protocol for  $\mathcal{R}$  is also called a *Proof of Knowledge (PoK)* for  $\mathcal{R}$ . Our  $\Sigma$ -protocol will have the peculiar property that the relation used to prove correctness turns out to be a subset of the one used to prove soundness. This will require extra care when proving security in Sect. 5.

**Special Honest Verifier Zero-Knowledge (SHVZK):** There exists a polynomial-time simulator that, given a statement  $x$  and a challenge  $\text{chall}$ , outputs a valid transcript  $(\text{com}, \text{chall}, \text{resp})$  that is indistinguishable from a real transcript.

**Definition 5.** A  $\Sigma$ -protocol  $(P_1, P_2, V)$  is computationally special honest verifier zero-knowledge if there exists a probabilistic polynomial time simulator  $\text{Sim}$  such that for all probabilistic polynomial time stateful adversaries  $\mathcal{A}$

$$\Pr \left[ \mathcal{A}(\text{com}, \text{chall}, \text{resp}) = 1 \mid \begin{array}{l} (x, w, \text{chall}) \leftarrow \mathcal{A}(1^\lambda); \\ \text{com} \leftarrow P_1(x, w); \\ \text{resp} = P_2(\text{chall}) \end{array} \right] \\ \approx \Pr \left[ \mathcal{A}(\text{com}, \text{chall}, \text{resp}) = 1 \mid \begin{array}{l} (x, w, \text{chall}) \leftarrow \mathcal{A}(1^\lambda); \\ (\text{com}, \text{resp}) \leftarrow \text{Sim}(x, \text{chall}) \end{array} \right].$$

If the above indistinguishability holds statistically against all unbounded adversaries  $\mathcal{A}$ , the protocol is said to be statistically SHVZK.

## 2.4 Non-Interactive Zero-Knowledge Proofs

In this paper, we consider non-interactive zero-knowledge (NIZK) proofs in the random oracle model that satisfy correctness, computational extractability and statistical zero-knowledge.

**Definition 6. (NIZK proofs.)** Let  $\mathcal{R}$  be a relation and let the language  $\mathcal{L}$  be a set of statements  $\{\text{st} \in \{0, 1\}^n\}$  such that for each statement  $\text{st} \in \mathcal{L}$ , there exists a corresponding witness  $\text{wit}$  such that  $(\text{st}, \text{wit}) \in \mathcal{R}$ . A non-interactive zero-knowledge (NIZK) proof system for  $\mathcal{R}$  is a tuple of probabilistic polynomial-time (PPT) algorithms  $\text{NIZK} = (P_{\text{NIZK}}, V_{\text{NIZK}})$  defined as follows (we assume that all algorithms in the description below have access to a common random oracle; we omit specifying it explicitly for ease of exposition):

- $P_{\text{NIZK}}(\text{st}, \text{wit})$ : A PPT algorithm that, given a statement  $\text{st} \in \{0, 1\}^n$  and a witness  $\text{wit}$  such that  $(\text{st}, \text{wit}) \in \mathcal{R}$ , outputs a proof  $\Pi$ .
- $V_{\text{NIZK}}(\text{st}, \Pi)$ : A deterministic algorithm that, given a statement  $\text{st} \in \{0, 1\}^n$  and a proof  $\Pi$ , either outputs 1 (accept) or 0 (reject).

The following correctness and security properties should be satisfied:

**Correctness.** For any  $(st, wit) \in \mathcal{R}$ , letting  $\Pi = P_{\text{NIZK}}(st, wit)$ , we must have  $V_{\text{NIZK}}(st, \Pi) = 1$ .

**Computational Extractability.** There exists an efficient PPT extractor  $\text{Ext}_{\text{NIZK}}$  such that for any security parameter  $\lambda \in \mathbb{N}$  and for any *polynomially bounded* cheating prover  $P^*$  where: (i)  $\text{Ext}_{\text{NIZK}}$  has rewinding access to  $P^*$ , and (ii)  $P_{\text{NIZK}}$ ,  $\text{Ext}_{\text{NIZK}}$  and  $P^*$  all have access to a common random oracle, letting  $(st, \Pi) \leftarrow P^*(1^\lambda)$  and  $wit = \text{Ext}_{\text{NIZK}}(st, \Pi)$ , if  $V_{\text{NIZK}}(st, \Pi) = 1$ , we must have  $\Pr[(st, wit) \in \mathcal{R}] > 1 - \text{negl}(\lambda)$ .

**Statistical Zero-knowledge.** There exists an efficient PPT simulator  $\text{Sim}_{\text{NIZK}}$  such that for any security parameter  $\lambda \in \mathbb{N}$  and for any non-uniform *unbounded* “cheating” verifier  $V^* = (V_1^*, V_2^*)$  where  $P_{\text{NIZK}}$ ,  $V_1^*$  and  $V_2^*$  all have access to a common random oracle, and such that  $\text{Sim}_{\text{NIZK}}$  is allowed programming access to the same random oracle, we have

$$\left| \Pr[V_2^*(st, \Pi, \xi) = 1 \wedge (st \in \mathcal{L})] - \Pr[V_2^*(st, \hat{\Pi}, \xi) = 1 \wedge (st \in \mathcal{L})] \right| \leq \text{negl}(\lambda),$$

where  $(st, wit, \xi) \leftarrow V_1^*(1^\lambda)$ ,  $\Pi \leftarrow P_{\text{NIZK}}(st, wit)$ , and  $\hat{\Pi} \leftarrow \text{Sim}_{\text{NIZK}}(st)$ .

### 3 Isogeny Graphs and Expansion

Let  $p$  be a prime and  $d$  an integer not divisible by  $p$ . An elliptic curve with level  $d$  Borel structure is a pair  $(E, C)$ , where  $E$  is an elliptic curve defined over a field of characteristic  $p$  and  $C$  is an order  $d$  cyclic subgroup of  $E[d]$ . We say that two such pairs  $(E_1, C_1)$  and  $(E_2, C_2)$  are isomorphic if there exists an isomorphism  $\phi : E_1 \rightarrow E_2$  such that  $\phi(C_1) = C_2$ . An automorphism of  $(E, C)$  is an isomorphism  $(E, C) \rightarrow (E, C)$ . They form the group  $\text{Aut}(E, C)$ .

Let  $\ell$  be a prime not dividing  $pd$ . The supersingular  $\ell$ -isogeny graph with level  $d$  structure  $G = G(p, d, \ell)$  is defined as follows. The set of vertices of  $G$  is a complete set  $V = V(p, d) = \{(E_i, C_i)\}$  of representatives of the set of isomorphism classes of supersingular elliptic curves with a level  $d$  Borel structure defined over  $\mathbb{F}_{p^2}$ . We note that each such class over  $\overline{\mathbb{F}_{p^2}}$  admits a model defined over  $\mathbb{F}_{p^2}$ : Each isomorphism class of supersingular elliptic curves has a representative  $E$  such that  $\#E(\mathbb{F}_{p^2}) = (p + 1)^2$  and thus the  $p^2$ -Frobenius acts as a scalar multiplication  $[-p]$ , so the kernel of any  $\ell$ -isogeny is  $\text{Gal}(\overline{\mathbb{F}_{p^2}})$ -invariant.

Now, the set of edges from  $(E, C)$  to  $(E', C')$  in  $G$  is the set of degree  $\ell$  isogenies from  $E$  to  $E'$  which map  $C$  to  $C'$ , modulo the action of  $\text{Aut}(E', C')$  (by postcomposition). The number of edges is independent of the representative of the isomorphism classes. When  $d = 1$ , we recover the usual definition of the supersingular  $\ell$ -isogeny graph.

This graph is directed. The out-degree of each vertex is  $\ell + 1$ , however the in-degree is not always  $\ell + 1$ , hence the adjacency matrix of the graph is not always symmetric.

### 3.1 Generalities on the Graph and Its Adjacency Matrix

Let  $V = \{(E_i, C_i)\}$  for  $i = 1, \dots, n$  be the vertex set of  $G = G(p, d, \ell)$ . On the complex vector space  $\mathbb{C}^V$ , we introduce the Hermitian form  $Q((E_i, C_i), (E_j, C_j)) = w_i \delta_{ij}$ , where  $\delta_{ij}$  is the Kronecker symbol and  $w_i := \frac{1}{2} |\text{Aut}(E_i, C_i)|$ . Denote by  $\|\cdot\|_Q$  the associated norm. We will compare  $\|\cdot\|_Q$  with the  $L^1$  and  $L^2$  norms on  $\mathbb{C}^V$ . The set  $\Omega$  of probability distributions on  $V$  is the set of vectors with real positive entries and  $L^1$  norm equal to 1. Consider also the vector  $\mathcal{E} = \sum_{i=1}^n \frac{1}{w_i} (E_i, C_i)$ , and  $s$  the probability distribution obtained normalizing  $\mathcal{E}$ . The following result contains a number of general facts about the adjacency matrix of  $G$ , which will be used later on.

- Theorem 7.** *1. The adjacency matrix  $A$  of  $G$  is self-adjoint with respect to  $Q$ ; in particular it is diagonalizable with real eigenvalues and eigenvectors;*  
*2. The vector  $\mathcal{E}$  is a left-eigenvector of eigenvalue  $\ell + 1$  of  $A$ ;*  
*3. The vector  $u$  with all entries equal to 1 is a right-eigenvector of  $A$ ; in particular its orthogonal complement  $S$  with respect to the  $L^2$  scalar product is preserved by right multiplication by  $A$ ;*  
*4.  $K := \inf\{\|v\|_Q : v \in \mathbb{C}^V \text{ and } \|v\|_{L^1} = 1\} = \left(\frac{(p-1)d}{12} \prod_q (1 + \frac{1}{q})\right)^{-1/2}$ , where the product index  $q$  runs over the prime divisors of  $d$ ;*  
*5.  $M := \sup\{\|\pi - s\|_Q : \pi \in \Omega\} \leq \sqrt{3}$ .*

*Proof.* The proof is given in the full version [5]. □

### 3.2 Proof of Theorem 3

We now prove that  $G = G(p, d, \ell)$  has the Ramanujan property. This follows from the first three items of Theorem 7 combined with the following result, whose proof heavily relies on the theory of modular forms. An immediate consequence is that  $G$  is connected and not bipartite, a different proof of which can be found in [39, Theorem 5.3.3].

**Theorem 8.** *Let  $S \subset \mathbb{C}^V$  be the subspace of vectors  $\sum_i v_i (E_i, C_i)$  such that  $\sum_i v_i = 0$ , as in Theorem 7. The eigenvalues of the action of  $A$  on  $S$  are all contained in the Hasse interval  $[-2\sqrt{\ell}, 2\sqrt{\ell}]$ .*

To prove Theorem 8, we assume standard notations and results about quadratic forms and modular forms, such as the ones from [31, 40, 53]. Given two elliptic curves with level structure  $(E_i, C_i)$  and  $(E_j, C_j)$ , we denote by  $\Lambda_{ij}$  the lattice of isogenies  $\phi: E_i \rightarrow E_j$  such that  $\phi(C_i) \subset C_j$ . The degree defines a quadratic form  $\text{deg}$  on  $\Lambda_{ij}$ . This quadratic module has rank four, level  $dp$  and determinant  $d^2 p^2$ . We can thus define the theta series

$$\Theta_{ij}(\tau) = \frac{1}{|\text{Aut}(E_j, C_j)|} \sum_{\phi \in \Lambda_{ij}} q^{\text{deg}(\phi)}, \quad \text{with } q = e^{2\pi i \tau}.$$

This function is in  $M_2(\Gamma_0(dp))$ , the space of modular forms of weight two for the modular group  $\Gamma_0(dp)$ , by [40, Theorem 4.2] (observe that in *loc. cit.* the

exponential is one because  $Q(h)$  is an integer; moreover, we choose  $P = 1$ ) or [53, Chapter IX, Theorem 5, page 218]. The above construction extends to an Hermitian pairing

$$\Theta: \mathbb{C}^V \otimes \mathbb{C}^V \rightarrow M_2(\Gamma_0(dp)) : ((\alpha_i)_i \otimes (\beta_j)_j) \mapsto \sum_{i,j} \alpha_i \beta_j \Theta_{ij}.$$

We call this pairing the Brandt pairing, even though there is a little ambiguity<sup>5</sup> in this set-up. The Brandt pairing is non-degenerate: let  $v = \sum c_i(E_i, C_i)$ , then the coefficient of  $q$  of  $\Theta(v, v)$  is the Hermitian norm of the vector of coefficients  $(\dots, c_i, \dots)$ . We will prove the following two key propositions.

**Proposition 9.** *The Brandt pairing intertwines the adjacency matrix  $A$  of  $G$  and the Hecke operator  $T_\ell$ ; in symbols  $T_\ell \Theta(w, v) = \Theta(wA, v)$  for all  $w, v \in \mathbb{C}^V$ .*

**Proposition 10.** *For every three elliptic curves with level structure  $(E_1, C_1)$ ,  $(E_2, C_2)$  and  $(E_3, C_3)$ , we have a cusp form*

$$\Theta((E_1, C_1), (E_3, C_3)) - \Theta((E_2, C_2), (E_3, C_3)).$$

The combination of these two results tells that the spectrum of the action of  $A$  restricted to  $S$  is contained into the spectrum of the action of the Hecke operator  $T_\ell$  on the space of cusp modular forms of weight two for  $\Gamma_0(dp)$ . The Ramanujan Conjecture, proved by Eichler, predicts that this second spectrum is contained in the Hasse interval, and hence proves Theorem 8.

We refer to [30, Theorem 8.2] for a proof of the Ramanujan Conjecture. In *loc. cit.* this result is proven only for eigenvectors of  $T_\ell$  which are new-forms. An eigenvector which is an old form will come from an embedding  $\iota: S_2(\Gamma_0(m)) \rightarrow S_2(\Gamma_0(dp))$  with  $m$  that divides  $dp$ . Since  $\ell$  is coprime with  $dp$ , the map  $\iota$  is  $T_\ell$ -equivariant (cf. [31, proof of Proposition 5.6.2]), so we can still deduce our result from [30, Theorem 8.2]. It is worth recalling that [30, Theorem 8.2] is stronger than what we need, as it applies to modular forms of every weight.

**Proof of Proposition 9.** We prove that both sides have the same  $q$ -expansions. For a power series  $F \in \mathbb{C}[[q]]$ , denote  $a_n(F)$  the coefficient of  $q^n$ . By definition

$$a_n(\Theta((E_i, C_i), (E_j, C_j))) = |\text{Aut}(E_j, C_j)|^{-1} \cdot |\text{Hom}^n((E_i, C_i), (E_j, C_j))|,$$

where  $\text{Hom}^n((E_i, C_i), (E_j, C_j))$  is the set of degree  $n$  isogenies in  $\Lambda_{ij}$ . For  $f \in M_2(\Gamma_0(dp))$ , we have  $a_n(T_\ell f) = a_{n\ell}(f) + \ell a_{n/\ell}(f)$  (see e.g. [31, Proposition 5.2.2]), where  $a_{n/\ell}(f)$  is set to zero in the case  $n/\ell \notin \mathbb{Z}$ . In particular,

$$\begin{aligned} a_n(T_\ell \Theta((E_i, C_i), (E_j, C_j))) &= \\ &= |\text{Aut}(E_j, C_j)|^{-1} \left( |\text{Hom}^{n\ell}((E_i, C_i), (E_j, C_j))| + \ell |\text{Hom}^{n/\ell}((E_i, C_i), (E_j, C_j))| \right) \end{aligned} \tag{2}$$

<sup>5</sup> Rather than using the condition  $\phi(C_i) \subset C_j$ , we could have defined  $\Lambda_{ij}$  using  $\phi(C_i) = C_j$ . The second definition does not give a lattice but still permits to define a pairing. This second pairing generalizes to all level structures, so it might deserve better the name of Brandt pairing. However, the second pairing gives a more complicated proof in the Borel case, so we have opted for the first one.

On the other side,

$$\begin{aligned}
 a_n(\Theta((E_i, C_i)A, (E_j, C_j))) &= \sum_C a_n(\Theta((E_i/C, \pi_C(C_i)), (E_j, C_j))) = \\
 &= |\text{Aut}(E_j, C_j)|^{-1} \sum_C |\text{Hom}^n((E_i/C, \pi_C(C_i)), (E_j, C_j))|
 \end{aligned}
 \tag{3}$$

where  $C$  varies among the cyclic non-trivial subgroups of  $E_i[\ell]$  of cardinality  $\ell$ , and  $\pi_C$  is the projection  $E_i \rightarrow E_i/C$ . For each  $C$  let

$$\begin{aligned}
 F_C: \text{Hom}^n((E_i/C, \pi_C(C_i)), (E_j, C_j)) &\longrightarrow \text{Hom}^{n\ell}((E_i, C_i), (E_j, C_j)) \\
 f &\longmapsto f \circ \pi_C,
 \end{aligned}$$

and let  $F$  be the disjoint union of the above maps. The map  $F$  is surjective: if  $\alpha: (E_i, C_i) \rightarrow (E_j, C_j)$  has degree  $n\ell$ , then  $\ker(\alpha) \cap E_i[\ell] \neq \{0\}$ , hence there exists a cyclic non-trivial  $C \subset \ker(\alpha) \cap E_i[\ell]$ , and we can write  $\alpha = f \circ \pi_C$ . In particular, let us compute the cardinality of the fiber  $F^{-1}(\alpha)$  for  $\alpha$  in the codomain. Each  $F_C$  is injective, hence  $|F^{-1}(\alpha)|$  is equal to the number of subgroups  $C$  such that  $F_C^{-1}(\alpha)$  is not empty, that is the number of subgroups  $C$  contained in  $\ker(\alpha) \cap E_i[\ell]$ . Hence

$$|F^{-1}(\alpha)| = \begin{cases} \ell + 1 & \text{if } \alpha = \ell\beta \text{ for some } \beta \in \text{Hom}^{n/\ell}((E_i, C_i), (E_j, C_j)), \\ 1 & \text{otherwise} \end{cases}$$

By (3), the domain of  $F$  has size exactly  $|\text{Aut}(E_j, C_j)| \cdot a_n(\Theta(A(E_i, C_i), (E_j, C_j)))$ , hence the proposition follows from (2) together with the above formula summed over  $\alpha$  in the codomain.  $\square$

**Proof of Proposition 10.** We have to show that, for any two pairs  $(E, C)$  and  $(E', C')$  and any cusp of  $X_0(dp)$ , the residue  $r$  of  $\Theta((E, C), (E', C'))d\tau$  does not depend on  $(E, C)$  and  $(E', C')$  at the cusp but only on  $p, d$  and the cusp.

By the discussion in [31, Section 3.8, page 103] each cusp can be represented as  $\begin{pmatrix} a \\ c \end{pmatrix}$  with  $c$  dividing  $dp$ , and  $r$  is equal to  $a_0(\Theta((E, C), (E', C'))|_M)$  for  $M$  any matrix in  $\text{SL}_2(\mathbb{Z})$  of the form  $\begin{pmatrix} a & b \\ c & \delta \end{pmatrix}$ .

By [53, Chapter IX, Equation (21), page 213], we have

$$r = \frac{1}{c^2pd} \sum_{\nu, \lambda \in A/cA} e\left(\frac{(a-1)\deg(\lambda) + \deg(\lambda + \nu) + (\delta-1)\deg(\nu)}{c}\right)$$

where  $e(z) = e^{2\pi iz}$ , and  $A$  is the lattice of isogenies from  $(E, C)$  to  $(E', C')$  which map  $C$  into  $C'$ . The above formula tells us that  $r$  only depends on  $M$  and on the quadratic form  $\deg: A/cA \rightarrow \mathbb{Z}/c\mathbb{Z}$ . Writing  $c = c_0p^\epsilon$  with  $c_0$  dividing  $N$  and  $\epsilon = 0, 1$  and using the Chinese remainder theorem we can split the quadratic form in two parts

$$A/cA = A/c_0A \times A/p^\epsilon A \xrightarrow{\deg \times \deg} \mathbb{Z}/c_0\mathbb{Z} \times \mathbb{Z}/p^\epsilon\mathbb{Z} \cong \mathbb{Z}/c\mathbb{Z}.$$

The quadratic module  $(\Lambda/c_0\Lambda, \text{deg})$  is (non-canonically) isomorphic to a Borel subalgebra of the algebra  $(\text{End}((\mathbb{Z}/c_0\mathbb{Z})^{\oplus 2}), \text{det})$ . An isomorphism can be obtained mapping it to  $\text{Hom}(E[c_0], E'[c_0])$ , and then choosing a symplectic basis.

If  $\epsilon = 0$  we are done, otherwise  $\epsilon = 1$ . Since  $[\text{Hom}(E, E') : \Lambda] = d$  is prime to  $p$ , we have  $\Lambda/p = \text{Hom}(E, E')/p = (\text{Hom}(E, E') \otimes \mathbb{Z}_p)/p$ , and the quadratic  $\mathbb{Z}_p$ -module  $\text{Hom}(E, E') \otimes \mathbb{Z}_p$  does not depend on the pair because, by the Deuring correspondence (see [57, Theorem 42.3.2.]) and by [57, Lemma 19.6.6], it is isomorphic to  $\lambda\mathcal{O}_p$  with the reduced norm, where  $\mathcal{O}_p$  is the maximal order in the non-ramified quaternions over  $\mathbb{Q}_p$ , and  $\lambda$  is an element of norm prime to  $p$ .  $\square$

### 3.3 Mixing Time of Non-backtracking Walks

We finally analyze the behavior of random walks in  $G = G(p, d, \ell)$ , which we will ultimately use to prove statistical indistinguishability of distribution arising from our proof of knowledge. First, observe that Theorem 7 item 2 shows that the probability distribution  $s$  introduced in Subsect. 3.1 is the stationary distribution on  $G$ . This is nearly the uniform distribution: all curves are equally likely, with the possible exception of the two curves with extra automorphisms,  $j = 1728$  and  $j = 0$ , which are respectively twice and thrice less likely.

We are going to determine the speed at which random walks converge to the stationary distribution. We focus on non-backtracking walks, which are the most useful for cryptographic protocols, but, because the graph is directed, we need some care to define them. Edges of  $G$  are equivalence classes of isogenies, so we choose a representative for each class. For an edge  $\alpha$  we define its dual edge as the chosen representative  $\beta$  for the class  $\text{Aut}(E, C)\hat{\alpha}$ , so that  $\beta\alpha = u\ell$  for  $u \in \text{Aut}(E, C)$ . Notice that the dual of  $\beta$  (as an edge) might be different from  $\alpha$ , but this is not relevant for us. We say that a random walk on  $G$  is non-backtracking walk if an edge is never followed by its dual.

With this “duality”, we have that isogenies of degree a power of  $\ell$  and with cyclic kernel (up to the equivalence  $\alpha \sim \beta$  iff  $\ker \alpha = \ker \beta$ ) correspond to non-backtracking walks.

**Theorem 11 (Mixing time).** *Let  $\pi$  be a probability distribution on  $G$ , and  $\pi^{(k)}$  the distribution obtained after a non-backtracking random walk of length  $k$ . Then we have*

$$d_{TV}(\pi^{(k)}, s) \leq \frac{1}{2}K^{-1}M \frac{(\ell + 1)(k + 1) - 2}{(\ell + 1)\sqrt{\ell^k}},$$

where  $K$  and  $M$  are as in Theorem 7 and  $d_{TV}$  denotes the total variation distance.

*Proof.* This follows from [2] for the case of undirected graphs. In the full version [5] we adapt the proof to the graph  $G(p, d, \ell)$ .  $\square$



## 4 Proof of Knowledge

Our goal is to provide a PoK of an isogeny walk  $\phi : E_0 \rightarrow E_1$  between two supersingular curves defined over  $\mathbb{F}_{p^2}$  that can be seamlessly plugged in a distributed SECUER generation protocol. For this, we need the following properties:

1. Compatible with any pair of curves  $(E_0, E_1)$ ; this rules out [36,37], which is restricted to a special starting curve  $E_0$ , and [24] and derivatives, which are restricted to curves defined over  $\mathbb{F}_p$ .
2. Statistically ZK, so that the security of the final SECUER does not hinge on computational assumptions brought in by the PoK; this rules out all other isogeny-based PoKs in the literature.
3. Post-quantum secure, possibly relying on as few additional assumptions as possible; this rules out many generic ZK proof systems.
4. Possibly compatible with any walk length and any base field  $\mathbb{F}_{p^2}$ .
5. Usable in practice for cryptographically sized finite fields.

Our new PoK inherits from the SIDH-based  $\Sigma$ -protocol of De Feo, Jao and Plût [25], and from the recent developments of De Feo, Dobson, Galbraith and Zobernig [23]. The common theme to all of them is to construct random SIDH squares (see (1)) on top of the secret isogeny  $\phi : E_0 \rightarrow E_1$  and to reveal some, but not all of the edges  $\psi, \psi', \phi'$  in response to a challenge. The reason these protocols are not statistically ZK is that the side  $\phi'$  is strongly correlated to the parallel side  $\phi$  (often unique given  $E_2$ ) and can thus easily be distinguished by an unbounded adversary.

Our first idea is to *make the walk  $\psi$  long enough* that the distribution of  $(E_2, \phi')$  becomes statistically close to the uniform distribution on supersingular curves with isogenies of degree  $\deg(\phi)$ . To prove it, we will use the properties of isogeny graphs with level structure analyzed in Sect. 3.

But making  $\psi$  longer is easier said than done. SIDH-based protocols are constrained in the lengths of  $\phi$  and  $\psi$  by the form of the prime  $p$ : typically,  $p + 1 = 2^a 3^b$  and then  $\deg(\phi) = 2^a$  and  $\deg(\psi) = 3^b$ . Our second idea is to *glue several SIDH squares together* to make longer walks (see Fig. 2). We call these larger diagrams *SIDH ladders*.

A valuable side-effect of gluing SIDH squares together is that we can free ourselves from the constraints on  $p$ . All we need is that isogenies of a small prime degree  $\ell$  coprime to  $\deg(\phi)$  can be computed efficiently, then we stack vertically sufficiently many SIDH squares to make  $\deg(\psi) = \ell^n$  as large as we need. In practice, we will take  $\deg(\phi) = 2^m$ ,  $\deg(\psi) = 3^n$ , and the protocol will be most efficient for SIDH primes, but in full generality our protocol works for any base field and any isogeny degree.

### 4.1 Protocol Description and Analysis

Let  $E_0, E_1$  be supersingular curves defined over a finite field  $\mathbb{F}_{p^2}$ , and let  $\phi : E_0 \rightarrow E_1$  be a cyclic separable isogeny of smooth degree  $d$ . Let  $\ell$  be a

small prime not dividing  $pd$ . Let  $C(m; r)$  be a statistically hiding and computationally binding commitment scheme. Our  $\Sigma$ -protocol is described in Fig. 1; it depends on a parameter  $n$ , controlling the length of the  $\ell$ -isogeny walks, that we will determine in Definition 15. The prover consists of two stateful algorithms  $(P_1, P_2)$ : the former is randomized and produces a commitment  $(\text{com}_2, \text{com}_3)$ , the latter receives a ternary challenge  $\text{chall} \in \{-1, 0, 1\}$  and produces a deterministic response  $\text{resp}$ . The verifier is a deterministic algorithm that receives  $((\text{com}_2, \text{com}_3), \text{chall}, \text{resp})$  and outputs a bit indicating whether or not the proof is accepted.

<p><math>P_1(E_0, E_1, \phi, n)</math>:</p> <ol style="list-style-type: none"> <li>1: Sample a random cyclic isogeny <math>\psi : E_0 \rightarrow E_2</math> of degree <math>\ell^n</math>;</li> <li>2: Construct the SIDH ladder <math>(E_0, E_1, E_2, E_3, \phi', \psi')</math> on <math>(\phi, \psi)</math>;</li> <li>3: Sample random strings <math>r_2, r_3</math>;</li> <li>4: <b>return</b> <math>(C(E_2; r_2), C(E_3; r_3))</math>.</li> </ol> <p><math>P_2(\text{chall})</math>:</p> <ol style="list-style-type: none"> <li>1: <b>if</b> <math>\text{chall} == -1</math> <b>then</b></li> <li>2:     <b>return</b> <math>(\psi, E_2, r_2)</math>;</li> <li>3: <b>else if</b> <math>\text{chall} == 1</math> <b>then</b></li> <li>4:     <b>return</b> <math>(\psi', E_3, r_3)</math>;</li> <li>5: <b>else if</b> <math>\text{chall} == 0</math> <b>then</b></li> <li>6:     <b>return</b> <math>(\phi', E_2, r_2, E_3, r_3)</math>.</li> </ol>	<p><math>V(E_0, E_1, d, n, (\text{com}_2, \text{com}_3), \text{chall}, \text{resp})</math>:</p> <ol style="list-style-type: none"> <li>1: <b>if</b> <math>\text{chall} == -1</math> <b>then</b></li> <li>2:     <math>(\psi, E_2, r_2) = \text{resp}</math>;</li> <li>3:     Check <math>\text{com}_2 = C(E_2; r_2)</math>;</li> <li>4:     Check <math>\psi</math> is an <math>\ell^n</math>-isogeny <math>E_0 \rightarrow E_2</math>;</li> <li>5: <b>else if</b> <math>\text{chall} == 1</math> <b>then</b></li> <li>6:     <math>(\psi', E_3, r_3) = \text{resp}</math>;</li> <li>7:     Check <math>\text{com}_3 = C(E_3; r_3)</math>;</li> <li>8:     Check <math>\psi'</math> is an <math>\ell^n</math>-isogeny <math>E_1 \rightarrow E_3</math>;</li> <li>9: <b>else if</b> <math>\text{chall} == 0</math> <b>then</b></li> <li>10:    <math>(\phi', E_2, r_2, E_3, r_3) = \text{resp}</math>;</li> <li>11:    Check <math>\text{com}_2 = C(E_2; r_2)</math>;</li> <li>12:    Check <math>\text{com}_3 = C(E_3; r_3)</math>;</li> <li>13:    Check <math>\phi'</math> is a cyclic <math>d</math>-isogeny <math>E_2 \rightarrow E_3</math>.</li> </ol>
---	--

**Fig. 1.** Interactive proof of knowledge of a cyclic isogeny  $\phi : E_0 \rightarrow E_1$  of degree  $d$ .

**Proposition 12.** *The  $\Sigma$ -protocol in Fig. 1 is correct for the relation*

$$\mathcal{R}_d = \{((E_0, E_1), \phi) \mid \phi : E_0 \rightarrow E_1 \text{ is a cyclic } d\text{-isogeny}\}.$$

*Assuming the commitment  $C$  is computationally binding, it is 3-special sound for the relation*

$$\mathcal{R}^* = \{((E_0, E_1), \chi) \mid \chi : E_0 \rightarrow E_1 \text{ is a cyclic } \ell^{2i}d\text{-isogeny for some } 0 \leq i \leq n\}.$$

*More precisely, there is a probabilistic polynomial time algorithm that, given three successful transcripts of the protocol with same commitments and distinct challenges, either recovers a witness  $\chi : E_0 \rightarrow E_1$ , or opens one of the commitments  $C(E_i; r_i)$  to two distinct values (breaking the binding property).*

*Proof. Correctness.* Suppose that the prover  $P = (P_1, P_2)$  and the verifier  $V$  follow the protocol. First note that, since the degree  $d$  of  $\phi$  is smooth, the SIDH ladder in  $P_1$  can be constructed as described in Sect. 4.2. Then it is clear that the commitments open successfully, and the verifier accepts the transcript for any challenge.

*3-Special Soundness.* Given three accepting transcripts  $(\text{com}, -1, \text{resp}_{-1})$ ,  $(\text{com}, 0, \text{resp}_0)$  and  $(\text{com}, 1, \text{resp}_1)$ , recover  $(\phi', E_2, r_2, E_3, r_3) = \text{resp}_0$  where  $\phi' : E_2 \rightarrow E_3$  is an isogeny. If the curves in  $\text{resp}_{-1}$  and  $\text{resp}_1$  are not equal to  $E_2$  and  $E_3$  respectively, then we can open one of the commitments  $\mathcal{C}(E_2; r_2)$  or  $\mathcal{C}(E_3; r_3)$  to two distinct outputs. Otherwise, we have  $\text{resp}_{-1} = (\psi, E_2, r_2)$  and  $\text{resp}_1 = (\psi', E_3, r_3)$  where  $\psi : E_0 \rightarrow E_2$  and  $\psi' : E_1 \rightarrow E_3$  are isogenies. Therefore  $\chi' = \widehat{\psi'} \circ \phi' \circ \psi$  is an isogeny from  $E_0$  to  $E_1$  of degree  $\ell^{2n}d$ . Factoring out the non-cyclic part of  $\chi'$ , we extract a cyclic isogeny  $\chi : E_0 \rightarrow E_1$  of degree  $\ell^{2i}d$  such that  $\chi' = [\ell^{2(n-i)}] \circ \chi$  for some  $0 \leq i \leq n$ ; however, like in the original SIDH PoK [23, 38], we cannot guarantee that  $i = 0$ .  $\square$

We are now going to define the simulator for proving ZK. Simulating  $\text{chall} = \pm 1$  is easy, however how well we can simulate the case  $\text{chall} = 0$  depends on the parameter  $n$  given to  $\text{P}_1$ . The opening  $(E_2, \phi' : E_2 \rightarrow E_3)$  can be equivalently viewed as the curve with level  $d$  Borel structure  $(E_2, \ker(\phi'))$ . Our goal is to have this opening distributed like a “random” vertex in the graph  $G = G(p, d, \ell)$ . To this effect, we define two sequences  $\mathcal{D}_1(k)$  and  $\mathcal{D}_2(k)$  of probability distributions on  $G$ , and we show that they converge as  $k$  grows.

**Definition 13.** Let  $\phi : E_0 \rightarrow E_1$  be a cyclic separable isogeny of degree  $d$ . Define

$$\begin{aligned} \mathcal{D}_1(k) &= \{(E_0/K, \tau(\ker(\phi)) \mid K \leftarrow \mathcal{C}_E(\ell^k), \tau : E_0 \rightarrow E_0/K\}, \\ \mathcal{D}_2(k) &= \{(E_0/K, C) \mid K \leftarrow \mathcal{C}_E(\ell^k), C \leftarrow \mathcal{C}_{E_0/K}(d)\}, \end{aligned} \quad (4)$$

where  $\mathcal{C}_E(f)$  is the uniform distribution on the cyclic subgroups of order  $f$  of  $E$ , up to  $\text{Aut}(E)$ .

**Lemma 14.** Keep notations as above, fix a positive real number  $\varepsilon$ , and let  $k$  be a positive integer such that

$$\tau(p, d, \ell, k) = \frac{1}{4}(p-1)^{1/2} \left(1 + \sqrt{d} \prod_{\substack{q|d \\ q \text{ prime}}} (1 + \frac{1}{q})^{1/2}\right) \cdot \left(k + \frac{\ell-1}{\ell+1}\right) \cdot \ell^{-k/2} \leq \varepsilon,$$

then  $d_{TV}(\mathcal{D}_1(k), \mathcal{D}_2(k)) \leq \varepsilon$ , where  $d_{TV}$  is the total variation distance between the two distributions, also known as statistical distance.

*Proof.* We bound the statistical distance of each of  $\mathcal{D}_1(k)$  and  $\mathcal{D}_2(k)$  from the stationary distribution of  $G(p, d, \ell)$ , as determined in Theorem 7, then we conclude with the triangle inequality. For  $\mathcal{D}_1(k)$ , we can directly apply Theorem 11. The argument for  $\mathcal{D}_2(k)$  is slightly more involved and is presented in the full version [5].  $\square$

**Definition 15.** Given  $p, d, \ell$  and  $m$ , define

$$n(p, d, \ell, m) = \min \{k \in \mathbb{Z} \mid \tau(p, d, \ell, k) \leq 2^{-m}\}.$$

**Proposition 16.** Let  $\lambda$  be a security parameter and let  $n = n(p, d, \ell, \lambda)$ . The  $\Sigma$ -protocol of Fig. 1 is statistically SHVZK for the relation  $\mathcal{R}_d$  defined in Proposition 12, assuming the commitment  $\mathcal{C}$  is statistically hiding.

*Proof.* We simulate the honest prover for each of the three challenges as follows.

chall = -1. Sample a random isogeny  $\psi : E_0 \rightarrow E_2$  of degree  $\ell^n$ , and random strings  $r_2, r_3$ . Set  $\text{com}_2 = C(E_2; r_2)$  and set  $\text{com}_3 = C(\perp; r_3)$ . Return  $(\text{com}_2, \text{com}_3), \text{chall}, (\psi, E_2, r_2)$ .

The isogeny  $\psi$  is distributed exactly like in the real protocol, thus this transcript is valid. Because  $C$  is statistically hiding, an adversary cannot distinguish  $\text{com}_3$  from a real commitment.

chall = 1. This is nearly identical to the above. The simulator samples  $\psi' : E_1 \rightarrow E_3$  of degree  $\ell^n$  and random strings  $r_2, r_3$ . It sets  $\text{com}_2 = C(\perp; r_2)$  and  $\text{com}_3 = C(E_3; r_3)$ , and returns  $(\text{com}_2, \text{com}_3), \text{chall}, (\psi', E_3, r_3)$ .

Because  $\ell$  is coprime to  $d$ , if  $\psi$  is uniformly distributed so is  $\psi'$ . Then, the transcript is indistinguishable from a real one as before.

chall = 0. Sample a random isogeny  $\psi : E_0 \rightarrow E_2$  of degree  $\ell^n$ , and then a random isogeny  $\rho : E_2 \rightarrow E_3$  of degree  $d$ . Sample random strings  $r_2, r_3$  and set  $\text{com}_2 = C(E_2; r_2)$  and  $\text{com}_3 = C(E_3; r_3)$ . Return  $(\text{com}_2, \text{com}_3), \text{chall}, (\rho, E_2, r_2, E_3, r_3)$ .

Thanks to Lemma 14, the statistical distance between the simulated  $(E_2, \ker(\rho))$  and  $(E_2, \psi(\ker(\phi)))$  is negligible. Because  $\rho$  is uniquely determined from  $\ker(\rho)$ , and the real response  $\phi'$  by  $\psi(\ker(\phi))$ , an adversary has negligible probability of distinguishing the transcript output by the simulator.  $\square$

## 4.2 Executing the Protocol

The protocol we just described crucially depends on the ability to construct a commutative square with sides of degrees  $d$  and  $\ell^n$ . The SIDH setting has  $p + 1 = d \cdot \ell^n$  so that the square can be constructed by simply pushing a single kernel point for  $\psi$  through  $\phi$  and vice versa. We refer to such a square as an *SIDH square*. For more general choices of  $\ell^n$  and  $d$ , the kernels are typically generated by points defined over very large extension fields, requiring superpolynomial space. We efficiently construct such “larger” squares by gluing together several SIDH squares in what we call *SIDH ladders*, as depicted in Fig. 2.

For simplicity, we shall present the case  $d = (2^a)^w$  and  $\ell^n = (3^b)^h$ , where  $2^a$  and  $3^b$  are the side lengths of an SIDH square, and  $w$  and  $h$  are positive integers defining the **width** and **height** of the ladders in units of SIDH squares. However, the technique generalizes easily to any coprime  $d$  and  $\ell^n$ , as long as isogenies of degrees  $d$  and  $\ell$  can be efficiently computed.

First, notice that there always exist some choice of  $a$  and  $b$  such that points (and hence kernel subgroups) of orders  $2^a$  and  $3^b$  can be represented efficiently. This is clear if the prime  $p$  is a SIDH prime where  $2^a 3^b \mid (p + 1)$ , but for a generic prime  $p$ , one can set  $a = b = 1$ : Points of order 2 and 3 are defined over a small extension field and can thus be efficiently represented. Moreover, any isogeny of degree  $(3^b)^h$  is the composition of  $h$  isogenies of degree  $3^b$  each, which can be stored as a sequence of  $h$  kernel generators which are efficiently representable.

This means that the prover can generate the isogeny  $\psi : E_0 \rightarrow E_2$  in step 2 of  $P_1$  by generating a random kernel  $K_{1,0}$  on  $E_0$ , computing the isogeny  $\psi_{1,0} :$

$E_0 \rightarrow E_0/\widehat{K_{1,0}} =: E_{1,0}$ , generating a random kernel  $K_{2,0}$  on  $E_{1,0}$  such that  $K_{2,0} \cap \ker \psi_{1,0} = \{0\}$  to prevent backtracking, and repeating the process  $h$  times to obtain a chain of  $h$  isogenies  $\psi_{i,0} : E_{i-1,0} \rightarrow E_{i,0}$ . The curve  $E_2$  is the codomain of the last isogeny  $\psi_{h,0}$ , i.e.,  $E_2 = E_{h,0}$ .

If the width  $w$  of the ladder is one, the prover can now recursively push the kernel  $G$  of the isogeny  $\phi = \phi_{0,1}$  through the isogenies  $\psi_{i,0}$  to obtain its image  $G_i$  on each curve  $E_{i,0}$ . Each horizontal isogeny  $\phi_{0,i}$  has kernel  $G_i$ , and the prover can compute the kernel of the right-side vertical isogeny  $\psi'_{i,0}$  as the image of the kernel of  $\psi_{i,0}$  under the isogeny  $\phi_{i-1,1}$ . Since each square composed of  $(E_{i,0}, E_{i+1,0}, E'_{i,0}, E'_{i+1,0})$  is a commutative diagram, so is the larger square  $(E_0, E_1, E_2, E_3)$ . In the general case where  $w > 1$ , the prover can use a similar approach for the horizontal isogeny  $\phi$  as used for the vertical isogeny  $\psi$ : The isogeny  $\phi$  can be written as the composition of  $w$  isogenies  $\phi_{0,w} \circ \dots \circ \phi_{0,1}$  of degree  $2^a$  and their kernels can be mapped through the vertical isogenies. In other words, the prover can glue horizontally  $w$  compatible ladders, one for each factor  $\phi_{0,i}$  of  $\phi$ . The right descending isogenies of each ladder are used as the left descending isogenies of the next one. This allows the prover to compute  $w \times h$  SIDH squares in such a way that the curves  $(E_0, E_1, E_2, E_3)$  and the isogenies between them form a commutative diagram. This is illustrated in Fig. 2. For the challenges  $\text{chall} = \pm 1$ , the prover reveals the isogenies  $\psi_{i,0}$  of the leftmost squares, or the isogenies  $\psi_{i,w}$  of the rightmost squares. For the challenge  $\text{chall} = 0$ , the prover responds with the isogenies  $\phi_{h,i}$  of the bottom squares.

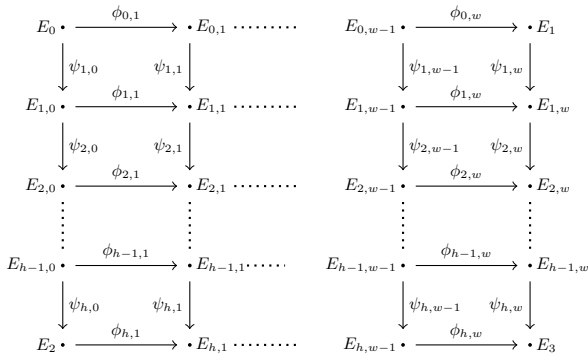


Fig. 2. An SIDH ladder.

Verification consists of evaluating (depending on the challenge) either  $w$  or  $h$  isogenies of degree  $2^a$  or  $3^b$ , which can be done efficiently. Generating the proof is slower, as the prover needs to fill in all the  $w \times h$  SIDH squares that make up the ladder. The proving complexity is thus quadratic in  $w$  and  $h$ , while the verification complexity is linear in  $w$  and  $h$ . However, the complexity of computing an SIDH square with degrees  $2^a$  or  $3^b$  is only quasilinear in  $a$  and  $b$  using sparse strategies [25]; thus, maximizing the size of SIDH squares improves

performance, which explains why SIDH primes are the most efficient scenario for this proof. If the degree of the isogenies and the size of the underlying field are kept constant, in the SIDH setting we have that  $2^a 3^b \mid (p + 1)$  for large values of  $a$  and  $b$  (in the order of several hundreds), and thus  $w$  and  $h$  can be small. For a generic prime, the prover might need to set  $a = b = 1$  and work with large values of  $w$  and  $h$ , incurring a quadratic cost, besides possibly having to compute points over an extension field of degree bounded by a small constant.

*Remark 17.* Above, we assumed that the degree of the witness  $\phi$  was  $d = (2^a)^w$ . As mentioned before, this can be generalized to any witness  $\phi$  of smooth degree  $d = d_1 \dots d_w$  as far as the  $d_i$ -torsion groups are accessible (ideally, one should have  $E_0[d_i] \subseteq E_0(\mathbb{F}_{p^2})$ ). In this case, one factors  $\phi$  as  $\phi = \phi_{0,w} \circ \dots \circ \phi_{0,1}$  where each isogeny  $\phi_{0,i}$  has degree  $d_i$ , and constructs compatible ladders for each  $\phi_{0,i}$ .

## 5 Distributed SECUER Setup and Its Security

In this section, we formally describe the distributed SECUER setup protocol and prove its security under a security definition using the simplified universal composability (SUC) framework due to Canetti, Cohen, and Lindell [13] in the real/ideal world paradigm. Our security definitions consider a *dishonest majority* corruption model, wherein the adversary can corrupt up to  $t - 1$  of the  $t$  participating parties in the distributed SECUER setup protocol. The protocol uses a non-interactive version of the  $\Sigma$ -protocol described in Sect. 4. We begin by formally describing this non-interactive zero-knowledge (NIZK) PoK protocol.

### 5.1 The NIZK Protocol

We transform the  $\Sigma$ -protocol of Sect. 4 into a NIZK using the standard Fiat-Shamir heuristic [33] for transforming interactive PoK protocols into NIZK proofs, albeit with the difference that soundness and zero-knowledge hold for slightly different languages.

**The NIZK Construction.** Let  $E_0, E_1$  be supersingular curves defined over a finite field  $\mathbb{F}_{p^2}$ , let  $\phi : E_0 \rightarrow E_1$  be a separable isogeny of smooth degree  $d$  and let  $C(m; r)$  be a statistically hiding and computationally binding commitment scheme. Additionally, let  $\Sigma = (P_1, P_2, V)$  be the interactive PoK protocol described in Sect. 4, let  $\lambda \in \mathbb{N}$  be the security parameter, let  $\ell$  be a small prime not dividing  $dp$ , let  $n = n(p, d, \ell, \lambda)$ , and let  $N = \text{poly}(\lambda)$  be a fixed polynomial. Finally, let  $H : \{0, 1\}^* \rightarrow \{-1, 0, 1\}^N$  be a random oracle. The NIZK proof system consists of a pair of algorithms  $\text{NIZK} = (P_{\text{NIZK}}, V_{\text{NIZK}})$  as described in Fig. 3. The prover algorithm  $P_{\text{NIZK}}$  is randomized and produces a proof  $\Pi$ . The verifier algorithm  $V_{\text{NIZK}}$  is deterministic; it receives the proof  $\Pi$  and outputs a bit  $b \in \{0, 1\}$  indicating whether or not the proof is accepted.

$P_{\text{NIZK}}(E_0, E_1, \phi, n, N)$ :

- 1: For each  $i \in [N]$ , sample  $(\text{com}_{2,i}, \text{com}_{3,i}) \leftarrow P_1(E_0, E_1, \phi, n)$ .
- 2: Set  $(\text{chall}_1, \dots, \text{chall}_N) = H((\text{com}_{2,1}, \text{com}_{3,1}), \dots, (\text{com}_{2,N}, \text{com}_{3,N}))$ .
- 3: For each  $i \in [N]$ , set  $\text{resp}_i = P_2(\text{chall}_i)$ .
- 4: **return**  $\Pi = (\{(\text{com}_{2,i}, \text{com}_{3,i}, \text{resp}_i)\}_{i \in [N]})$ .

$V_{\text{NIZK}}(E_0, E_1, \Pi, N)$ :

- 1: Parse  $\Pi$  as  $(\{(\text{com}_{2,i}, \text{com}_{3,i}, \text{resp}_i)\}_{i \in [N]})$ .
- 2: Compute  $(\text{chall}_1, \dots, \text{chall}_N) = H((\text{com}_{2,1}, \text{com}_{3,1}), \dots, (\text{com}_{2,N}, \text{com}_{3,N}))$ .
- 3: For each  $i \in [N]$ , compute  $b_i = V(E_0, E_1, (\text{com}_{2,i}, \text{com}_{3,i}), \text{chall}_i, \text{resp}_i)$ .
- 4: Output  $b = \wedge_{i \in [N]} b_i$ .

**Fig. 3.** The NIZK.

**Correctness, Extractability and ZK.** Correctness follows immediately from the correctness of the underlying  $\Sigma$ -protocol. We state and prove the following propositions for extractability and ZK.

**Proposition 18.** *Assuming that  $\Sigma = (P_1, P_2, V)$  satisfies 3-special soundness with respect to the relation  $\mathcal{R}^*$  (described in Proposition 12) and that  $H$  is a random oracle, the NIZK  $\text{NIZK} = (P_{\text{NIZK}}, V_{\text{NIZK}})$  satisfies extractability (and hence soundness) with respect to the relation  $\mathcal{R}^*$ .*

*Proof.* We provide an informal proof overview. We begin by noting that  $\Sigma$  is a public-coin protocol, and that there exists a probabilistic polynomial-time algorithm that extracts a witness from 3 accepting transcripts corresponding to  $N$  parallel executions of  $\Sigma$  w.r.t. the same statement. Consequently, we can invoke the generalized forking lemma of [11] to argue the existence of a probabilistic polynomial-time witness-extraction algorithm for NIZK. This completes the proof of extractability (and hence, soundness) for NIZK.  $\square$

**Proposition 19.** *Assuming that  $\Sigma = (P_1, P_2, V)$  is statistically SHVZK for the relation  $R_d$  (described in Proposition 16) and that  $H$  is a random oracle, the NIZK  $\text{NIZK} = (P_{\text{NIZK}}, V_{\text{NIZK}})$  is statistically ZK for the relation  $R_d$ .*

*Proof.* We again provide an informal proof overview. Let  $\text{Sim}_\Sigma$  be a ZK simulator that simulates an accepting transcript for the underlying  $\Sigma$ -protocol (as described in the proof of ZK for  $\Sigma$ ). We construct a ZK simulator  $\text{Sim}_{\text{NIZK}}$  that simulates an accepting proof as follows:

1.  $\text{Sim}_{\text{NIZK}}$  simulates the random oracle  $H$  as follows: it maintains a local table consisting of tuples of the form  $(x, y) \in \{0, 1\}^* \times \{-1, 0, 1\}^N$ . On receiving a query  $x \in \{0, 1\}^*$  from the adversary  $\mathcal{A}$ , it looks up this table to check if an entry of the form  $(x, y)$  exists. If yes, it responds with  $y$ . Otherwise, it responds with a uniformly sampled  $y \leftarrow \{-1, 0, 1\}^N$ , and programs the random oracle as  $H(x) := y$  by adding the entry  $(x, y)$  to the table.
2. For each  $i \in [N]$ ,  $\text{Sim}_{\text{NIZK}}$  internally invokes the simulator  $\text{Sim}_\Sigma$  for the underlying  $\Sigma$ -protocol to obtain the  $i$ -th accepting transcript of the form

$$((\text{com}_{2,i}, \text{com}_{3,i}), \text{chall}_i, \text{resp}_i).$$

3. At this point,  $\text{Sim}_{\text{NIZK}}$  aborts if the adversary  $\mathcal{A}$  has already issued a random oracle query on the input  $x = ((\text{com}_{2,1}, \text{com}_{3,1}), \dots, (\text{com}_{2,N}, \text{com}_{3,N}))$ .
4. Otherwise,  $\text{Sim}_{\text{NIZK}}$  programs the random oracle as

$$H((\text{com}_{2,1}, \text{com}_{3,1}), \dots, (\text{com}_{2,N}, \text{com}_{3,N})) := (\text{chall}_1, \dots, \text{chall}_N),$$

and outputs the simulated proof as  $\Pi = (\{(\text{com}_{2,i}, \text{com}_{3,i}, \text{resp}_i)\}_{i \in [N]})$ .

We note that  $\text{Sim}_{\text{NIZK}}$  runs in polynomial time as long as  $\text{Sim}_\Sigma$  runs in polynomial time. Additionally, if  $\text{Sim}_{\text{NIZK}}$  does not abort, it outputs a simulated proof that is distributed in a statistically indistinguishable manner from the distribution of a real proof, assuming that  $\text{Sim}_\Sigma$  outputs a simulated accepting transcript with distribution statistically indistinguishable from a real accepting transcript for  $\Sigma$ . Finally,  $\text{Sim}_{\text{NIZK}}$  aborts with only negligible probability, since the adversary  $\mathcal{A}$  guesses  $((\text{com}_{2,i}, \text{com}_{3,i}), \text{chall}_i, \text{resp}_i)$  for each  $i \in [n]$  with at most negligible probability. This completes the proof of statistical ZK for NIZK.  $\square$

### 5.2 Our Distributed SECUER setup protocol

We now move to the distributed SECUER setup protocol. Let  $P_1, \dots, P_t$  be a set of  $t$  participating parties and let  $E_0$  be some fixed starting curve. In a nutshell, the idea is to have the parties act sequentially: each  $P_i$  at its own turn performs a secret random walk  $E_{i-1} \rightarrow E_i$  and broadcasts  $E_i$  and a NIZK PoK of the secret walk. We claim that, as long as one party is honest, the final curve  $E_t$  is a SECUER.

To get any security guarantee, we need to carefully set the parameters of the random walk  $E_{i-1} \rightarrow E_i$ . The natural choice is to fix some small prime  $q$ , not dividing  $\ell p$ , and to take a random walk long enough that the distribution of  $E_i$  is negligibly far from the stationary distribution on the  $q$ -isogeny graph  $G(p, 1, q)$ . For example we may set  $q = 2$  and  $\ell = 3$ , then Theorem 11 provides a precise bound to set the length  $\delta = n(p, 1, q, \lambda)$  of the  $q$ -walk as a function of the security parameter, and ultimately the parameter  $n(p, q^\delta, \ell, \lambda)$  of the PoK.

*Remark 20.* For increased efficiency, we may choose to perform shorter  $q$ -walks  $E_{i-1} \rightarrow E_i$  of length  $\log_q(p)$ . This length approximates the diameter of the supersingular  $q$ -isogeny graph; hence, it ensures that the secret isogeny can reach almost any curve in the graph.

Under mild assumptions, this choice would still yield a secure protocol, but it would also make the security proof somewhat more involved. For this reason, we shall stick here to the more conservative choice of walking long enough to ensure nearly stationary distribution of  $E_i$ .

We formally describe the protocol (referred to as  $\Gamma_{\text{SECUER}}$  henceforth). Assume that  $E_0$  is known to all the parties at the start. Let  $\text{NIZK} = (\text{P}_{\text{NIZK}}, \text{V}_{\text{NIZK}})$  be the non-interactive proof as described above. The protocol  $\Gamma_{\text{SECUER}}$  proceeds in  $t$  rounds while only using broadcast channels of communication, where round- $i$  for each  $i \in [t]$  is as follows:



- Party  $P_i$  performs a  $q$ -isogeny walk starting at curve  $E_{i-1}$  and ending at curve  $E_i$  (where  $E_{i-1}$  and  $E_i$  are both supersingular curves defined over  $\mathbb{F}_{p^2}$ ), such that party  $P_i$  knows a separable isogeny  $\phi_i : E_{i-1} \rightarrow E_i$  of degree  $q^\delta$ , where  $\delta = n(p, 1, q, \lambda)$ .
- Party  $P_i$  generates  $\Pi_i \leftarrow \text{P}_{\text{NIZK}}(E_{i-1}, E_i, \phi_i, n, N)$ , where  $n = n(p, q^\delta, \ell, \lambda)$ , and broadcasts  $(E_i, \Pi_i)$  to all other parties.
- Each party  $P_j$  for  $j \in [t] \setminus \{i\}$  verifies the NIZK proof  $\Pi_i$  by computing  $b_i = \text{V}_{\text{NIZK}}(E_{i-1}, E_i, \Pi_i, N)$ . If  $b_i = 0$  (i.e., the proof is invalid),  $P_j$  aborts.

At the end of round- $t$ , all parties output  $E_t$  to be the final output curve.

**Correctness.** Correctness of  $\Gamma_{\text{SECUER}}$  follows immediately from the correctness guarantees of the NIZK.

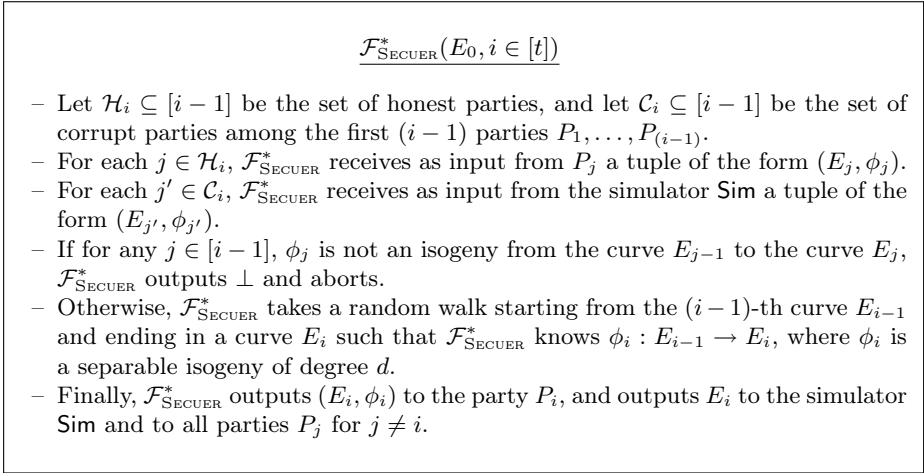
### 5.3 Proof of Security for $\Gamma_{\text{SECUER}}$

We now present the proof of security for  $\Gamma_{\text{SECUER}}$  using the simplified universal composability (SUC) framework [13] in the real/ideal world paradigm. We consider a *dishonest majority* corruption model, wherein the adversary can corrupt up to  $(t - 1)$  of the  $t$  participating parties.

**The Ideal Functionality.** Intuitively, the ideal functionality for distributed SECUER setup should simply take as input the initial curve  $E_0$  and output a SECUER  $E_t$ . It is however not obvious how to model the property of being a SECUER in the plain SUC model: a game based definition, stating that an adversary who can compute  $\text{End}(E_t)$  can be used to break some other assumption, appears to be more appropriate.

Thus, we prove security in two steps. First, we prove that  $\Gamma_{\text{SECUER}}$  securely emulates a *less-than-ideal* functionality  $\mathcal{F}_{\text{SECUER}}^*$  (described in Fig. 4) that enforces that: (a) for each  $i \in [t]$ , if a corrupt party  $P_i$  outputs a curve  $E_i$ , it must know a valid isogeny  $\phi_i : E_{i-1} \rightarrow E_i$ , and (b) for each  $i \in [t]$ , if an honest party  $P_i$  outputs a curve  $E_i$ , then the corresponding isogeny  $\phi_i : E_{i-1} \rightarrow E_i$  is hidden from the adversary. This step relies on the extractability and ZK properties of the NIZK protocol described above. Next, we prove that, assuming the hardness of the endomorphism ring problem in the  $\mathcal{F}_{\text{SECUER}}^*$ -hybrid model, the output curve  $E_t$  is a SECUER, i.e. that the (malicious) adversary cannot compute  $\text{End}(E_t)$ .

**Theorem 21.** *Assuming that  $\text{NIZK} = (\text{P}_{\text{NIZK}}, \text{V}_{\text{NIZK}})$  satisfies extractability and zero-knowledge, and assuming the hardness of the endomorphism ring problem (Definition 1) and GRH, the output  $E_t$  of the protocol  $\Gamma_{\text{SECUER}}$  is a SECUER if at least one party  $P_{i^*}$  for some  $i^* \in [t]$  is honest.*



**Fig. 4.** The Ideal functionality  $\mathcal{F}_{\text{SECUER}}^*$

**Secure Emulation of  $\mathcal{F}_{\text{SECUER}}^*$ .** We now prove that  $\Gamma_{\text{SECUER}}$  securely emulates the less-than-ideal functionality  $\mathcal{F}_{\text{SECUER}}^*$ . Our proof is in the real/ideal world paradigm defined formally as follows.

*The Real World.* The following entities engage in the real protocol  $\Gamma_{\text{SECUER}}$ : (i) a set  $\mathcal{H} \subseteq [t]$  of honest parties, (ii) a real-world adversary  $\mathcal{A}$  controlling a set  $\mathcal{C} \subset [t]$  of corrupt parties, and (iii) the environment  $\mathcal{E}$  that provides  $E_0$  to each party, interacts with the real-world adversary  $\mathcal{A}$ , receives the final output curve  $E_t$  from the honest parties, and eventually outputs a bit  $b \in \{0, 1\}$ .

*The Ideal World.* The following entities interact with the functionality  $\mathcal{F}_{\text{SECUER}}^*$ : (i) A set  $\mathcal{H} \subseteq [t]$  of honest parties, where for each  $i \in \mathcal{H}$ , party  $P_i$  directly forwards its secret isogeny to  $\mathcal{F}_{\text{SECUER}}^*$ , (ii) an ideal-world simulator  $\text{Sim}$  that sends inputs to  $\mathcal{F}_{\text{SECUER}}^*$  on behalf of a set  $\mathcal{C} \subset [t]$  of corrupt parties, and (iii) the environment  $\mathcal{E}$  that provides each party with the starting curve  $E_0$ , interacts with the simulator  $\text{Sim}$ , receives the final output curve  $E_t$  from the functionality, and eventually outputs a bit  $b \in \{0, 1\}$ .

For any  $t$ -party SECUER setup protocol  $\Gamma_{\text{SECUER}}$ , any adversary  $\mathcal{A}$ , any simulator  $\text{Sim}$ , and any environment  $\mathcal{E}$ , we define the following random variables:

- $\text{real}_{\Gamma_{\text{SECUER}}, \mathcal{A}, \mathcal{E}}$ : denotes the output of the environment  $\mathcal{E}$  after interacting with the adversary  $\mathcal{A}$  during a real-world execution of  $\Gamma_{\text{SECUER}}$ .
- $\text{ideal}_{\mathcal{F}_{\text{SECUER}}^*, \text{Sim}, \mathcal{E}}$ : denotes the output of the environment  $\mathcal{E}$  after interacting with the simulator  $\text{Sim}$  in the ideal world.

**Theorem 22.** *Assuming that  $\text{NIZK} = (\text{P}_{\text{NIZK}}, \text{V}_{\text{NIZK}})$  satisfies extractability and zero-knowledge, for any security parameter  $\lambda \in \mathbb{N}$  and any probabilistic polynomial time (PPT) adversary  $\mathcal{A}$ , there exists a PPT simulator  $\text{Sim}$  such that, for*

any PPT environment  $\mathcal{E}$ , we have

$$\left| \Pr [\text{real}_{\Gamma_{\text{SECUER}}, \mathcal{A}, \mathcal{E}} = 1] - \Pr [\text{ideal}_{\mathcal{F}_{\text{SECUER}}^*, \text{Sim}, \mathcal{E}} = 1] \right| \leq \text{negl}(\lambda).$$

*Proof.* We prove this theorem by constructing a PPT simulator  $\text{Sim}$  that simulates the view of the environment  $\mathcal{E}$  in the ideal world. Details are given in the full version [5].  $\square$

**Analyzing  $E_t$  in  $\mathcal{F}_{\text{SECUER}}^*$ -hybrid Model.** Based on the above secure emulation guarantee, we now analyze the output  $E_t$  of  $\Gamma_{\text{SECUER}}$  in the  $\mathcal{F}_{\text{SECUER}}^*$ -hybrid model. Concretely, we state and prove the following theorem.

**Theorem 23.** *Assuming the hardness of the endomorphism ring problem and GRH, the output  $E_t$  of  $\mathcal{F}_{\text{SECUER}}^*(E_0, t)$  is a SECUER if at least one party is honest.*

To prove this theorem, we first prove the following lemma.

**Lemma 24.** *Assuming the hardness of the endomorphism ring problem, the output  $E_i$  of  $\mathcal{F}_{\text{SECUER}}^*(E_0, i)$  for  $i \in [t]$  is a SECUER whenever  $P_i$  is honest.*

*Proof.* Suppose that there exists an adversary  $\mathcal{A}$  corrupting a dishonest majority of the parties that efficiently computes the endomorphism ring of  $E_i$  with non-negligible probability. Also assume that  $\mathcal{A}$  corrupts all of  $P_1, \dots, P_{i-1}$ . We can use  $\mathcal{A}$  to construct an algorithm  $\mathcal{B}$  that solves the endomorphism ring problem. The algorithm  $\mathcal{B}$  receives as input a uniformly random curve  $E^*/\mathbb{F}_{p^2}$ , internally runs the adversary  $\mathcal{A}$  to emulate the outputs of the corrupt parties  $P_1, \dots, P_{i-1}$ , and finally feeds  $\mathcal{A}$  with  $E_i := E^*$ . The view of the adversary  $\mathcal{A}$  is properly simulated by  $\mathcal{B}$ , since  $E_i$  output by  $\mathcal{F}_{\text{SECUER}}^*$  and  $E^*$  provisioned by  $\mathcal{B}$  are statistically indistinguishable (here we use Theorem 11, which crucially follows from the honest party taking a  $q$ -walk of length  $n(p, 1, q, \lambda)$ ). Finally,  $\mathcal{B}$  uses  $\mathcal{A}$  to recover the endomorphism ring of  $E^*$  with non-negligible probability. This concludes the proof of Lemma 24.  $\square$

We now prove Theorem 23. We break the proof into two cases: (i) when  $P_t$  is honest, and (ii) when  $P_t$  is corrupt. The proof for case (i) is immediate from Lemma 24. Hence, we focus on case (ii). Let  $\mathcal{H} \subseteq [t]$  be the set of honest parties, and let  $i^* = \max(\{i : P_i \in \mathcal{H}\})$ . By Lemma 24,  $E_{i^*}$  must be a SECUER. Now, suppose that  $E_t$  is not a SECUER, i.e., there exists an adversary  $\mathcal{A}$  corrupting dishonest majority of the parties that efficiently computes the endomorphism ring of  $E_t$  with non-negligible probability. Since all of  $P_{i^*+1}, \dots, P_t$  are corrupt,  $\mathcal{A}$  knows a walk from  $E_{i^*}$  to  $E_t$  in the  $\ell$ -isogeny graph. However, since  $E_t$  is not a SECUER,  $\mathcal{A}$  can use the reduction [59] (assuming GRH) to recover  $\text{End}(E_{i^*})$ , thereby violating Lemma 24. This completes the proof of Theorem 23.  $\square$

Finally, the proof of Theorem 21 follows immediately from the proofs of Theorem 22 and Theorem 23, which completes the proof of security for our distributed SECUER setup protocol  $\Gamma_{\text{SECUER}}$ .

**Table 1.** Parameters and corresponding secret/proof size for each of the four SIKE finite fields.

		Degree		SIDH Squares		Size (kB)	
$\log(p)$	Reps	2-isog	3-isog	Columns	Rows	Secret	Proof
434	219	705	890	4	7	0.99	191.19
503	219	774	977	4	7	1.13	215.75
610	329	1010	1275	4	7	1.39	404.32
751	438	1280	1616	4	7	1.69	662.63

## 6 Implementation and Results

In this section, we report on our proof-of-concept implementation of our proof of knowledge (Sect. 4), including a discussion of proof sizes and running times. Moreover, we lay out concretely how one may deploy the trusted setup protocol from Sect. 5 in the real world.

**Parameter Selection.** The base-field primes  $p$  in our proof-of-knowledge implementation are taken from the four SIKE parameter sets **p434**, **p503**, **p610**, and **p751**. As discussed in Sect. 4.2, our proof of knowledge achieves its optimal efficiency for SIDH-style primes. Moreover, those primes have been featured extensively in the literature, and thus appear to be the obvious choice to demonstrate our proof of knowledge. That said, we stress once more that our techniques are generic and can be applied in any choice of characteristic.

We use the degree  $q = 2$  for the random walks  $E_i \rightarrow E_{i-1}$ , and  $\ell = 3$  for the random walks of the  $\Sigma$ -protocol of Fig. 1. Like Sect. 5, we set  $\delta = n(p, 1, 2, \lambda)$  for the length of the 2-walks, and  $n = n(p, 2^\delta, 3, \lambda)$  for the 3-walks. Lastly, the  $\Sigma$ -protocol needs to be repeated several times to achieve a negligible soundness error. Since one repetition has soundness error  $2/3$ , the protocol needs to be repeated  $-\lambda/\log(2/3)$  times to achieve  $2^{-\lambda}$  soundness error. We target the same security levels as the corresponding SIKE parameter sets, i.e.,  $\lambda = 128$  for **p434** and **p503**,  $\lambda = 192$  for **p610**, and  $\lambda = 256$  for **p751**. The resulting conservative parameters are summarized in Table 1.

**Implementation.** We developed an optimized implementation<sup>6</sup> of our proof of knowledge (Sect. 4.1) for the trusted-setup application (Sect. 5) based on version 3.5.1 of Microsoft’s SIDH library<sup>7</sup>. Our implementation inherits and benefits from all lower-level optimizations contained in that library, and it supports a wide range of platforms with optimized code for a variety of Intel and ARM processors. Compiling our software produces two command-line tools **prove** and **verify**, which use a simple ASCII-based interface to communicate the data contributed to the trusted setup.

<sup>6</sup> The source code is available at <https://github.com/trusted-isogenies/SECUER-pok>.

<sup>7</sup> <https://github.com/microsoft/PQCrypto-SIDH>.

The implementation closely follows the strategy outlined in Sect. 4.2. This includes the choices  $d = (2^a)^w$  and  $\ell^n = (3^b)^h$ ; thus, both the witness and the commitment isogenies are uniformly random cyclic isogenies of degree  $d$  and  $\ell^n$  respectively. To reduce latency, we additionally exploit parallelism: Recall that the proof of knowledge is repeated many times to achieve a low soundness error; indeed most of the computations are independent between those repetitions and can thus easily be performed at the same time on a multi-core system. This is confirmed by experimental results, where our implementation is observed to parallelize almost perfectly when run on an eight-core processor.

Sampling purely random large-degree isogenies with code from SIDH comes with two caveats: First, the sampling of “small” squares must avoid backtracking between the individual squares being glued to ensure that the composition is cyclic in the end; in both cases this is done by keeping track of the kernel of the dual of the last prime-degree step of the previous square and avoiding points lying above this “forbidden” kernel when choosing the next square. Besides that, the specific isogeny formulas used in SIDH fail for the 2-torsion point  $(0, 0)$ , which can be resolved by changing to a different Montgomery model each time this kernel point is encountered. For curves revealed in the proof, the choice of Montgomery model should be randomized to avoid leakage. Similarly, the kernel generators of the horizontal isogeny  $\phi'$  also need to be randomized, as Lemma 14 only distinguishes cyclic subgroups and revealing specific generators may leak.

Our software sacrifices some performance for simplicity, which aids auditability and hence helps increase trust in the results of a trusted-setup ceremony. Some unused optimizations: Two-isogenies are faster to compute than three-isogenies, and since the SIDH ladder is taller than wider, swapping the role of two- and three-isogenies in the trusted-setup application could somewhat improve the resulting performance. For simplicity, our implementation also only uses full SIDH squares, and thus all isogeny degrees are rounded up to the closest multiple of an SIDH square; shortening the sides of some of the squares can save time. We also did not apply all optimizations to reduce the proof size. This includes applying SIDH-style compression techniques [20] to the points contained in the proof, cutting their size approximately in half. Moreover, applying a slight bias when sampling the challenges  $\text{chall}_i$  means smaller responses can appear more often, at the expense of requiring slightly more repetitions; we investigated this tradeoff and determined that the potential improvement is essentially void.

**Results.** We benchmarked the three algorithms (instance generation, proving, and verification) that make up the zero-knowledge proof of knowledge. We run our tests on an ARM Apple M1 Pro with eight cores, and we averaged the running times of 100 iterations for the parallel implementation and the running times of 50 iterations of the single-core version. The resulting timings are shown in Table 2. They demonstrate that the algorithm is highly practical and can realistically be used within a trusted setup protocol: Generating proofs of knowledge for all four base fields takes less than five core-minutes on a modern CPU. Note that these algorithms need to be run only once per contributor.

**Table 2.** Benchmarks for instance generation, proving, and verification of our proof of isogeny knowledge for each of the four SIKE finite fields.

$\log(p)$	Single-core Time (s)			Eight-core Time (s)		
	Instance	Prove	Verify	Instance	Prove	Verify
434	0.01	18.15	1.93	0.01	2.96	0.32
503	0.01	25.70	2.71	0.01	4.17	0.44
610	0.02	74.82	7.69	0.02	12.12	1.24
751	0.04	162.47	17.01	0.04	26.07	2.89

**Real-World Deployment.** We briefly discuss how we intend to deploy the trusted setup protocol proposed in Sect. 5. The goals of such a deployment include include a transparent setup that allows parties to trust the process, a low bar of entry to participate in the protocol, and a secure system that can withstand Sybil and Denial-of-Service (DoS) attacks.

Firstly, we are releasing at <https://github.com/trusted-isogenies/> a set of tools that participants can download and run to generate a valid addition to the trusted setup, and for ceremony orchestrators to validate protocol submissions on the server-side. To increase user trust, we also provide higher-level versions (e.g., in SageMath) of some components. Moreover, the proof format is made public, so that any party can—if they choose to—re-implement the algorithms and generate a compatible proof.

Then, we propose leveraging the existing infrastructure of git and GitHub to host our distributed protocol. Thus, each party  $E_i$  can generate a random walk from the latest curve  $E_{i-1}$  to a new curve  $E_i$ , generate a PoK of their secret isogeny walk, and submit the new curve and the PoK to the server as a pull request (PR). The server is a separate git repository and execution environment maintaining the sequence of curves and the proofs, with checks that are run automatically against submissions from parties. The repository automation verifies that the submitted PoK of the isogeny between the current curve  $E_{i-1}$  at the end of the walk (the ‘tip’ curve) and the new proposed curve  $E_i$  is valid, and that the PR does not rewrite any previous history. If the checks pass, the PR is rebased on top of the main branch, adding the new PoK of the latest hop, and updating the tip curve to  $E_i$ . New parties in the protocol will generate isogeny walks starting from the new tip curve.

If the chain of isogenies diverges, i.e. if some party submits a new curve and PoK starting from a curve other than the tip, the new submission is rejected. This may happen when several parties try to contribute at the same time. To minimize the amount of wasted prover work, we parallelize verification and reject invalid proofs as early as possible.

The configuration for the continuous integration checks is maintained in a separate repository to prevent modification from protocol parties. Hosting the protocol on GitHub raises the bar to Sybil attacks, as it requires all parties to have a GitHub account with a verified email address. Using our tool requires

generation of a GitHub personal access token to authenticate when generating the submission, which further complicates automation/collusion.

The end result of the protocol is a public git repository whose final commit contains a sequence of curves and valid PoKs of isogenies between them, the last of which is the final SECUER  $E_t$ , a curve with unknown endomorphism ring, in a parsable hex encoding. Anyone can pull down this artifact and verify the sequence of curves and proofs independently if they wish.

## 7 Conclusion

In this work, we analyzed a distributed SECUER generation protocol, and proposed a concrete instantiation with strong security guarantees based on a novel proof of isogeny knowledge. To demonstrate the practical feasibility of our protocol, we are going to run a distributed SECUER generation ceremony, scaling to hundreds of participants, using the technology outlined in Sect. 6.

Our new PoK is especially well-suited for SIDH-like base fields, but can be used reasonably well with fields  $\mathbb{F}_{p^2}$  of any characteristic. Generic ZK proof systems, such as the SumCheck protocol used in [18], would be an alternative to our PoK. After this work was published, Cong, Lai and Levin [19] designed an R1CS encoding of 2-isogeny walks that they fed to various generic proof systems. Their results show that Aurora [6], in particular, can be quite competitive, giving a measurable speed boost at the cost of a moderate increase in proof size. Currently, the question of which proof system to use appears to be context-dependent.

None of the currently known techniques are particularly well suited for proving knowledge of an isogeny walk over  $\mathbb{F}_p$ : our PoK and generic proof systems are much more efficient when the walks consist of isogenies of small degree such as 2 or 3, which is not possible over  $\mathbb{F}_p$ . SeaSign-like techniques [24, 29] are at least one order of magnitude slower than our PoK, and scale much worse. CSI-FiSh [8] is reasonably efficient, but limited to the base field of CSIDH-512. We think generating SECUERS over  $\mathbb{F}_p$  efficiently is an interesting open problem.

To show the security of the proof of knowledge, we developed the theory of supersingular isogeny graphs with level structure, in particular proving that they possess the Ramanujan property. In this work we only focused on the so-called Borel level structure, however similar properties can be proven for more general level structures. In a follow-up work, we will develop the general theory of these graphs, prove bounds on their eigenvalues, and discuss consequences for isogeny-based cryptography.

**Acknowledgments.** We are grateful to the reviewers and to Shai Levin for helping catch several mistakes and misprints. We thank Jeff Burdges for valuable discussions during the preparation of this work.

## References

1. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 411–439. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-64834-3\\_14](https://doi.org/10.1007/978-3-030-64834-3_14)
2. Alon, N., Benjamini, I., Lubetzky, E., Sodin, S.: Non-backtracking random walks mix faster. *Commun. Contemp. Math.* **9**(4), 585–603 (2007). <https://doi.org/10.1142/S0219199707002551>
3. Arpin, S.: Adding level structure to supersingular elliptic curve isogeny graphs (2022). <https://doi.org/10.48550/ARXIV.2203.03531>, [arXiv:2203.03531](https://arxiv.org/abs/2203.03531)
4. Basso, A.: A post-quantum round-optimal oblivious PRF from isogenies. *Cryptology ePrint Archive, Paper 2023/225* (2023). <https://eprint.iacr.org/2023/225>
5. Basso, A., et al.: Supersingular curves you can trust. *Cryptology ePrint Archive, Report 2022/1469* (2022). <https://eprint.iacr.org/2022/1469>
6. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17653-2\\_4](https://doi.org/10.1007/978-3-030-17653-2_4)
7. Bernstein, D.J., De Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. *Open Book Series* **4**(1), 39–55 (2020). <https://doi.org/10.2140/obs.2020.4.39>
8. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. LNCS, vol. 11921, pp. 227–247. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-34578-5\\_9](https://doi.org/10.1007/978-3-030-34578-5_9)
9. Boneh, D., Kogan, D., Woo, K.: Oblivious pseudorandom functions from isogenies. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 520–550. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-64834-3\\_18](https://doi.org/10.1007/978-3-030-64834-3_18)
10. Booher, J., et al.: Failing to hash into supersingular isogeny graphs. *Cryptology ePrint Archive, Report 2022/518* (2022). <https://eprint.iacr.org/2022/518>
11. 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, Part II. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49896-5\\_12](https://doi.org/10.1007/978-3-662-49896-5_12)
12. Burdges, J., De Feo, L.: Delay encryption. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 302–326. Springer, Heidelberg (2021). [https://doi.org/10.1007/978-3-030-77870-5\\_11](https://doi.org/10.1007/978-3-030-77870-5_11)
13. Canetti, R., Cohen, A., Lindell, Y.: A simpler variant of universally composable security for standard multiparty computation. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 3–22. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48000-7\\_1](https://doi.org/10.1007/978-3-662-48000-7_1)
14. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH (preliminary version). *Cryptology ePrint Archive, Report 2022/975* (2022). <https://eprint.iacr.org/2022/975>
15. 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, Part III. LNCS, vol. 11274, pp. 395–427. Springer, Heidelberg (2018). [https://doi.org/10.1007/978-3-030-03332-3\\_15](https://doi.org/10.1007/978-3-030-03332-3_15)



16. Castryck, W., Panny, L., Vercauteren, F.: Rational isogenies from irrational endomorphisms. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 523–548. Springer, Heidelberg (2020). [https://doi.org/10.1007/978-3-030-45724-2\\_18](https://doi.org/10.1007/978-3-030-45724-2_18)
17. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. *J. Cryptol.* **22**(1), 93–113 (2007). <https://doi.org/10.1007/s00145-007-9002-x>
18. Chávez-Saab, J., Rodríguez-Henríquez, F., Tibouchi, M.: Verifiable isogeny walks: Towards an isogeny-based postquantum VDF. In: AlTawy, R., Hülsing, A. (eds.) SAC 2021. LNCS, vol. 13203, pp. 441–460. Springer, Heidelberg (2022). [https://doi.org/10.1007/978-3-030-99277-4\\_21](https://doi.org/10.1007/978-3-030-99277-4_21)
19. Cong, K., Lai, Y.F., Levin, S.: Efficient isogeny proofs using generic techniques. Cryptology ePrint Archive, Report 2023/037 (2023). <https://eprint.iacr.org/2023/037>
20. Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J., Urbanik, D.: Efficient compression of SIDH public keys. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 679–706. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-319-56620-7\\_24](https://doi.org/10.1007/978-3-319-56620-7_24)
21. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006). <https://eprint.iacr.org/2006/291>
22. De Feo, L., et al.: Seta: Supersingular encryption from torsion attacks. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part IV. LNCS, vol. 13093, pp. 249–278. Springer, Heidelberg (2021). [https://doi.org/10.1007/978-3-030-92068-5\\_9](https://doi.org/10.1007/978-3-030-92068-5_9)
23. De Feo, L., Dobson, S., Galbraith, S.D., Zobernig, L.: SIDH proof of knowledge. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 310–339. Springer, Heidelberg (2022). [https://doi.org/10.1007/978-3-031-22966-4\\_11](https://doi.org/10.1007/978-3-031-22966-4_11)
24. De Feo, L., Galbraith, S.D.: SeaSign: Compact isogeny signatures from class group actions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part III. LNCS, vol. 11478, pp. 759–789. Springer, Heidelberg (2019). [https://doi.org/10.1007/978-3-030-17659-4\\_26](https://doi.org/10.1007/978-3-030-17659-4_26)
25. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.* **8**(3), 209–247 (2014). <https://doi.org/10.1515/jmc-2012-0015>
26. De Feo, L., Kieffer, J., Smith, B.: Towards practical key exchange from ordinary isogeny graphs. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 365–394. Springer, Heidelberg (2018). [https://doi.org/10.1007/978-3-030-03332-3\\_14](https://doi.org/10.1007/978-3-030-03332-3_14)
27. De Feo, L., Kohel, D., Leroux, A., Petit, C., Wesolowski, B.: SQISign: compact post-quantum signatures from quaternions and isogenies. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part I. LNCS, vol. 12491, pp. 64–93. Springer, Heidelberg (2020). [https://doi.org/10.1007/978-3-030-64837-4\\_3](https://doi.org/10.1007/978-3-030-64837-4_3)
28. De Feo, L., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. LNCS, vol. 11921, pp. 248–277. Springer, Heidelberg (2019). [https://doi.org/10.1007/978-3-030-34578-5\\_10](https://doi.org/10.1007/978-3-030-34578-5_10)
29. Decru, T., Panny, L., Vercauteren, F.: Faster SeaSign signatures through improved rejection sampling. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019. pp. 271–285. Springer, Heidelberg (2019). [https://doi.org/10.1007/978-3-030-25510-7\\_15](https://doi.org/10.1007/978-3-030-25510-7_15)



30. Deligne, P.: La conjecture de Weil : I. Publications Mathématiques de l’IHÉS **43**, 273–307 (1974). [http://www.numdam.org/item/PMIHES\\_1974\\_\\_43\\_\\_273\\_0/](http://www.numdam.org/item/PMIHES_1974__43__273_0/)
31. Diamond, F., Shurman, J.: A First Course in Modular Forms, Graduate Texts in Mathematics, vol. 228. Springer-Verlag, New York (2005). <https://doi.org/10.1007/978-0-387-27226-9>
32. Eisenträger, K., Hallgren, S., Lauter, K.E., Morrison, T., Petit, C.: Supersingular isogeny graphs and endomorphism rings: reductions and solutions. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 329–368. Springer, Heidelberg (2018). [https://doi.org/10.1007/978-3-319-78372-7\\_11](https://doi.org/10.1007/978-3-319-78372-7_11)
33. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO’86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). [https://doi.org/10.1007/3-540-47721-7\\_12](https://doi.org/10.1007/3-540-47721-7_12)
34. Fouotsa, T.B., Kutas, P., Merz, S.P., Ti, Y.B.: On the isogeny problem with torsion point information. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part I. LNCS, vol. 13177, pp. 142–161. Springer, Heidelberg (2022). [https://doi.org/10.1007/978-3-030-97121-2\\_6](https://doi.org/10.1007/978-3-030-97121-2_6)
35. 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, Part I. LNCS, vol. 10031, pp. 63–91. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53887-6\\_3](https://doi.org/10.1007/978-3-662-53887-6_3)
36. 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, Part I. LNCS, vol. 10624, pp. 3–33. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-319-70694-8\\_1](https://doi.org/10.1007/978-3-319-70694-8_1)
37. Galbraith, S.D., Petit, C., Silva, J.: Identification protocols and signature schemes based on supersingular isogeny problems. *J. Cryptol.* **33**(1), 130–175 (2019). <https://doi.org/10.1007/s00145-019-09316-0>
38. Ghantous, W., Pintore, F., Veroni, M.: Collisions in supersingular isogeny graphs and the SIDH-based identification protocol. *Cryptology ePrint Archive*, Report 2021/1051 (2021). <https://eprint.iacr.org/2021/1051>
39. Goren, E.Z., Kassaei, P.L.:  $p$ -adic dynamics of Hecke operators on modular curves. *Journal de Théorie des Nombres de Bordeaux* **33**(2), 387–431 (2021). <https://www.jstor.org/stable/48618785>
40. Hijikata, H., Pizer, A.K., Shemanske, T.R.: The basis problem for modular forms on  $\Gamma_0(N)$ . *Mem. Amer. Math. Soc.* **82**(418), vi+159 (1989). <https://doi.org/10.1090/memo/0418>
41. Jao, D., et al.: SIKE. Tech. rep., National Institute of Standards and Technology (2020). <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>
42. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.Y. (ed.) Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011. pp. 19–34. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-25405-5\\_2](https://doi.org/10.1007/978-3-642-25405-5_2)
43. Kohel, D.: Endomorphism rings of elliptic curves over finite fields. Ph.D. thesis, University of California at Berkeley (1996). <https://www.i2m.univ-amu.fr/perso/david.kohel/pub/thesis.pdf>
44. Lai, Y.F., Galbraith, S.D., de Saint Guilhem, C.: Compact, efficient and UC-secure isogeny-based oblivious transfer. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 213–241. Springer, Heidelberg (2021). [https://doi.org/10.1007/978-3-030-77870-5\\_8](https://doi.org/10.1007/978-3-030-77870-5_8)

45. Love, J., Boneh, D.: Supersingular curves with small noninteger endomorphisms. *Open Book Series* **4**(1), 7–22 (2020). <https://doi.org/10.2140/obs.2020.4.7>
46. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A direct key recovery attack on SIDH. In: To appear in EUROCRYPT 2023. LNCS, Springer, Heidelberg (2023). <https://eprint.iacr.org/2022/1026>
47. Mestre, J.F.: La méthode des graphes. Exemples et applications. In: Proceedings of the international conference on class numbers and fundamental units of algebraic number fields (Katata, 1986). Nagoya University, Nagoya (1986). <https://wstein.org/msri06/refs/mestre-method-of-graphs/mestre-fr.pdf>
48. Mula, M., Murru, N., Pintore, F.: Random sampling of supersingular elliptic curves. *Cryptology ePrint Archive, Report 2022/528* (2022). <https://eprint.iacr.org/2022/528>
49. Petit, C.: Faster algorithms for isogeny problems using torsion point images. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part II. LNCS, vol. 10625, pp. 330–353. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-319-70697-9\\_12](https://doi.org/10.1007/978-3-319-70697-9_12)
50. Pizer, A.K.: Ramanujan graphs and Hecke operators. *Bulletin of the American Mathematical Society (N.S.)* **23**(1) (1990). <https://doi.org/10.1090/S0273-0979-1990-15918-X>
51. de Quehen, V., et al.: Improved torsion-point attacks on SIDH variants. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 432–470. Springer, Heidelberg, Virtual Event (2021). [https://doi.org/10.1007/978-3-030-84252-9\\_15](https://doi.org/10.1007/978-3-030-84252-9_15)
52. Robert, D.: Breaking SIDH in polynomial time. *Cryptology ePrint Archive, Report 2022/1038* (2022). <https://eprint.iacr.org/2022/1038>
53. Schoeneberg, B.: Elliptic modular functions: an introduction. *Die Grundlehren der mathematischen Wissenschaften, Band 203*, Springer, Heidelberg (1974). <https://doi.org/10.1007/978-3-642-65663-7>
54. Sterner, B.: Commitment schemes from supersingular elliptic curve isogeny graphs. *Math. Cryptol.* **1**(2), 40–51 (2022). <https://journals.flvc.org/mathcryptology/article/view/130656>
55. Stolbunov, A.: Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. Math. Commun.* **4**(2), 215–235 (2010). <https://doi.org/10.3934/amc.2010.4.215>
56. Vélou, J.: Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris, Séries A* **273**, 305–347 (1971)
57. Voight, J.: Quaternion algebras, *Graduate Texts in Mathematics*, vol. 288. Springer, Cham (2021). <https://doi.org/10.1007/978-3-030-56694-4>
58. Wesolowski, B.: Orientations and the supersingular endomorphism ring problem. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 345–371. Springer, Heidelberg (2022). [https://doi.org/10.1007/978-3-031-07082-2\\_13](https://doi.org/10.1007/978-3-031-07082-2_13)
59. Wesolowski, B.: The supersingular isogeny path and endomorphism ring problems are equivalent. In: 62nd FOCS. pp. 1100–1111. IEEE Computer Society Press (2022). <https://doi.org/10.1109/FOCS52979.2021.00109>



# On Valiant’s Conjecture

## Impossibility of Incrementally Verifiable Computation from Random Oracles

Mathias Hall-Andersen  and Jesper Buus Nielsen <sup>(✉)</sup> 

Aarhus University, Aarhus, Denmark  
jbn@cs.au.dk

**Abstract.** In his landmark paper at TCC 2008 Paul Valiant introduced the notion of “incrementally verifiable computation” which enables a prover to incrementally compute a succinct proof of correct execution of a (potentially) long running process. The paper later won the 2019 TCC test of time award. The construction was proven secure in the random oracle model without any further computational assumptions. However, the overall proof was given using a non-standard version of the random-oracle methodology where sometimes the hash function is a random oracle and sometimes it has a short description as a circuit. Valiant clearly noted that this model is non-standard, but conjectured that the standard random oracle methodology would not suffice. This conjecture has been open for 14 years. We prove that if the proof system can receive a long witness as input in an incremental manner and is also zero-knowledge then the conjecture is true. Valiant’s original construction does not have these properties but can easily be extended to have them in his model. We relate our result to recent possibility and impossibility results for SNARKs and incrementally verifiable computation.

**Keywords:** Idealized Models · Lower Bounds · Separations and Impossibility Results · Proof Systems · Zero-Knowledge

## 1 Introduction

*Incrementally Verifiable Computation.* In his landmark paper Paul Valiant [21] introduced the notion of “incrementally verifiable computation” (IVC) which enables a prover to incrementally compute a succinct proof of correct execution of a (potentially) long running process. At any time the prover can suspend the computation and return a proof of correct execution leading up to the present state. This paper inspired a lot of later constructions, including modern recursive SNARK constructions, and won the 2019 TCC test-of-time award.

---

M. Hall-Andersen—Funded by the Concordium Foundation.

J.B. Nielsen—Partially funded by The Concordium Foundation; The Danish Independent Research Council under Grant-ID DFF-8021-00366B (BETHE); The Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

© International Association for Cryptologic Research 2023

C. Hazay and M. Stam (Eds.): EUROCRYPT 2023, LNCS 14005, pp. 438–469, 2023.

[https://doi.org/10.1007/978-3-031-30617-4\\_15](https://doi.org/10.1007/978-3-031-30617-4_15)

The methodology applied by Valiant is incremental. The computation applies the same step function  $T$  a number of  $\ell$  times. There is an initial state  $M_0$  and  $M_i = T(M_{i-1})$ . There is also an initial proof  $\pi_0$ , the empty string say. To construct the proof  $\pi_i$  that  $M_i = T^i(M_0)$  one constructs a proof of knowledge of  $(M_{i-1}, \pi_{i-1})$  for which it holds that  $M_i = T(M_{i-1})$  and that  $\pi_{i-1}$  verifies the statement  $M_{i-1} = T^{i-1}(M_0)$ .<sup>1</sup>

The proofs are *succinct* in the sense that their size depend only poly-logarithmically on the number of steps  $i$ . Verification time also depends only poly-logarithmically on  $i$ . So neither the prover nor the verifier can just rerun the computation from  $M_0$ . Note that some notion of succinctness must follow from any reasonable notion of incrementality. Otherwise each new proof could just be recomputed from  $M_0$ , which hardly qualifies as “incremental”.

The soundness of the recursive proof system is proven in the random oracle model without any further computational assumptions. However, Valiant need to apply a non-standard version of the random oracle model. When proving soundness of the proof system extending a proof by one step it is assumed that the hash function is a random oracle. However, when recursively proving that  $\pi_{\ell-1}$  verifies it is assumed that the hash function has a short description as a circuit. This gives a somewhat interesting model where the hash function at different times has contradicting properties. The paper is very up front about this and justifies it by the conjecture that it seems that the standard random oracle methodology is not enough:

... When we try to recursively embed this system the recursion breaks down because, even at the first level of recursion, we are no longer trying to prove statements about classical computation but rather statements of the form “ $M$  with oracle access to  $\mathcal{O}$  accepts the following string...”  
*Thus standard applications of random oracles do not appear to help.* [our emphasis]. ...

–Paul Valiant [21]

In [8] Chiesa and Liu show impossibility results for proofs in relativized worlds, i.e., proofs of exactly the form “ $M$  with oracle access to  $\mathcal{O}$  accepts the following string...” They show that  $\text{DTIME}(t)^\mathcal{O} \not\subseteq \text{PCP}(o(t), o(t))^\mathcal{O}$  and  $\text{NTIME}(t)^\mathcal{O} \not\subseteq \text{PCP}(\text{poly}(t), o(t))^\mathcal{O}$ , which can informally be interpreted as not *all* statements of the form “ $M$  with oracle access to  $\mathcal{O}$  accepts the following string...” can have a non-trivial proof where not all the oracle queries of  $M$  are checked by the verifier. But if the verifier checks all oracle queries of the prover and each steps makes just one query then the verifier is not succinct. As noted in [8] this “gives strong evidence that Valiant’s approach was in some sense justified.” However, it does not conclusively rule out that Valiant’s approach can be instantiated in

<sup>1</sup> As detailed later this description is oversimplified but will suffice for our discussion. The real recursive strategy is more involved to tame the complexity of knowledge extraction.

the standard random oracle model. It cannot be ruled out that a proof system can be constructed where the verifier is simple enough that it does not fall prey to the Chiesa-Liu results, as they only prove that not *all* statements have such a proof.

And even if we could rule out the explicitly *recursive* strategy, where we extend a proof by proving knowledge of an accepting sub-proof, then it might still be possible to do *incremental* proofs in the random oracle model using some other strategy. In particular, the end result of Valiant’s approach is to give a proof about random oracle devoid computation, which is not ruled out by the Chiesa-Liu results. As already noted by Valiant:

... It remains an interesting question whether the goals of this paper may be attained in some other way using random oracles. ...

–Paul Valiant [21]

In the present paper we show that Valiant was correct and that indeed the standard random oracle model is not sufficient for incremental proofs. We rule out not just explicitly recursive designs, but general incremental designs. As we discuss below we do not prove impossibility for the exact setting studied by Valiant: we need to assume two additional but natural properties of the proof system, which Valiant’s construction can easily be extended to have.

*Non-deterministic Computation.* The first additional assumption we need is that the ongoing computation can receive a long witness as input in an incremental manner. The verifier is assumed to only have access to a short instance. In a modern setting this could be a verifier knowing only the genesis block and a recent block of a blockchain and the prover wants to succinctly prove that the blockchain has some property, like the verifier having been paid a certain amount defined by the overall activity on the blockchain. Here the genesis block plus the recent block is the short instance and the blockchain is the long witness. It is a natural question whether the proof can be computed incrementally, say by consuming the blockchain block-by-block.

The original notion of IVC considers only deterministic computation: the verifier is provided with a Turing machine and the prover convinces the verifier that the provided state is reached after executing the Turing machine for some number of steps. Motivated by “distributed computation” Chiesa and Tromer [10] subsequently generalized IVC to the powerful notion of “Proof-Carrying Data” (PCD) in which the correct computation of a function taking multiple inputs can be proven given proofs of correctness for each of the inputs, e.g., the computation of  $F(G_1(w_1), G_2(w_2))$  can be proven given  $y_1 = G_1(w_1)$ ,  $y_2 = G_2(w_2)$  and corresponding proofs-of-knowledge  $\pi_1, \pi_2$  for  $w_1, w_2$ . For our impossibilities we use the abstraction of “non-deterministic incrementally verifiable computation”, which is a special case of PCD with “fan-in” 1 with the same function applied in each step, or equivalently, a generalization of IVC where each step of the Turing machine may take a witness.

*Zero-Knowledge via Reprogramming the Random Oracle.* Our impossibility results also assume that the incremental proof is zero-knowledge. Succinct arguments already information theoretically hides most of the witness, as the proof is much shorter than the witness. For general PCD zero-knowledge is even a natural requirement: different steps of the computation may be performed by mutually distrustful parties which do not want to share their secrets.

The notion of ZK which we consider is as follows. The simulator is given a correct state and proof for step  $i$  and must then produce a proof for step  $i + 1$  without knowing the corresponding witness. This simulated proof should look indistinguishable to a PPT adversary. The simulator may inspect and reprogram the random oracle, but to make its job harder we give the adversary access to querying the random oracle *before* the simulation is made. This can be seen as giving a limited form of auxiliary information on the oracle to the adversary. It is discussed by Goldreich in [15] why auxiliary information is important for composability of ZK proofs. As discussed by Unruh in [20] it is also essential for composability to give the adversary auxiliary information on the random oracle. Otherwise the security assumption assumes the random oracle appears magically *after* the adversary specified its strategy. In this case each proof would need its own fresh random oracle even for sequential composition. For the case of IVC this would require that a fresh random oracle appears after each proof step, which is a very unnatural model.

We note that our impossibilities hold under any computational assumptions, as long as the ZK simulation proceeds only by reprogramming the random oracle. It therefore does not, e.g., rule out constructions from common reference strings where the simulation relies on the trapdoor of the CRS.

*In This Paper.* In this paper the main result is to show that succinct, zero-knowledge non-deterministic IVC from random oracles is impossible in the following two cases.

1. There exist collision intractable hash functions and the proof system has *knowledge soundness*. Knowledge soundness and zero-knowledge may depend on standard model computational assumptions including non-falsifiable assumptions.
2. There exist perfectly binding rerandomizable commitments and the proof system has *soundness*. Both soundness and zero-knowledge may depend on standard-model computation assumptions including non-falsifiable assumptions.

*Universal Knowledge Soundness.* For the first result we consider a notion of knowledge extraction with a universal extractor: we require that there exists a poly-time extractor which works for all poly-time adversaries. The extractor is given the code of the adversary as input, so it can still use *non-blackbox extraction*. However, quantifying the extractor before the adversary makes it hard to use for instance knowledge-of-exponent assumptions or any other assumption of the form “for all adversaries there exists an extractor such that ...”.

We note that the first result still stands if one makes knowledge assumptions or, in general, any non-falsifiable assumptions. The only restriction we make is that our definition of universal knowledge soundness makes it harder to exploit these assumptions.

We now give an overview of our proof techniques and discuss the results in more detail, discuss generalisations, and compare to existing (im)possibility results for PCDs and SNARKs.

*PCD via Recursion.* Above we discussed recursive proofs as being simply sequential. To avoid confusion let us note that in Valiant’s original paper [21] IVC is constructed using a tree of linear-time extractable CS proofs [19] in which the leafs each prove a step of the execution, while the parents (a CS proof) proves the correct execution of the verifier on the two children (CS proofs) which each cover half of the computation time. By maintaining just  $\log(T)$  such proofs the computation can be extended in the obvious way. The tree structure is essential to ensuring polynomial-time extraction, since the linear-time knowledge extractor need only be applied  $\log(T)$  times recursively to extract the entire computational trace. In later works [3, 4] from zk-SNARKS the proofs are composed iteratively, which implies that the proof as far as we know only is sound for computation of constant depth. Lately, in practical schemes/deployments, the efficiency of the recursive extraction is largely ignored: instead showing that a single level of recursion is extractable [5, 6]. Common to all known constructions is the non-blackbox use of (parts of) the verifier.

*Incremental PCD.* Our results apply not only to recursive proofs but to succinct incremental proofs in general. We look at incremental proofs produced by some  $\ell$  number of succinct steps. By succinct we mean that the state of the prover passed on from one step to the next has size  $\text{poly}(|\mathcal{R}|, \lambda, \log \ell)$ , where  $\mathcal{R}$  is the PPT relation checking that one step was computed correctly,  $\lambda$  is the security parameter, and  $\ell$  is the number of steps. Each computation of a proof need not be state bounded, only the state passed on to the next step. We also require that the verifier has running time  $\text{poly}(|\mathcal{R}|, \lambda, \log \ell)$ .

*Technical Overview.* We sketch the main ideas behind the impossibility results. For all results the witness for an  $\ell$ -step proof is a long random vector  $\vec{w} = (w_1, \dots, w_\ell)$ , where  $w_i$  is given (only) in step  $i$ . Each  $w_i$  is security parameter long. We first prove that no adversary (cheating prover) can produce an accepting proof if there is some index  $i$  such that we do not give it the witness  $w_i$  used in iteration  $i$ . We sketch why this is true.

For the case of collision intractable hash functions the computation computes a Merkle-Damgård hash of  $\vec{w}$ , consuming one  $w_i$  per step. If the prover could succeed in producing an accepting proof without using  $w_i$ , then we could apply the knowledge extractor to the accepting proof and recover  $w_i$ . It is easy to see that this can be used to violate collision intractability.

For the case of perfectly binding rerandomizable commitments the instance is a long sequence of commitments  $c_0, c_1, \dots$  where the claim is that the sequence



was produced as a sequence of rerandomisations of the previous commitment. Step  $i$  of the proof gets as input  $c_{i-1}$  and  $c_i$  and the witness is the randomness used to produce  $c_i$  as a rerandomisation of  $c_{i-1}$ . The commitment  $c_0$ , of step 1, is a commitment of 0. By perfect binding it follows that for true instances the commitment  $c_\ell$  of step  $\ell$  is also a commitment of 0. The missing witness will now be the randomness used for a rerandomisation in some step  $i$ . If the prover is not given this randomness it cannot distinguish a commitment  $c_i$  of 0 from a commitment of 1. Hence we can do a switch from a commitment  $c_{i-1}$  of 0 to a commitment  $c_i$  of 1. So if for a true instance the prover could succeed without  $w_i$  then it could also succeed for a false instance, breaking soundness.

We then finish the proofs by showing that if the verifier does not make  $\Theta(\ell)$  queries to the random oracle then there exists an adversary producing an accepting proof and which does *not* use all witnesses, giving a contradiction.

This proof only uses that there is a zero-knowledge simulator in the random oracle model: it can simulate a given step if allowed to reprogram the random oracle. The indistinguishability of the real view and the simulated view may depend on other computational assumptions. For each step  $m$  and each step  $n > m$  we use that the simulator works by programming the oracle to argue that the verifier of step  $n$  must make a check *related to* the proof of step  $m$ . To see this note that if we simulate step  $m$  and the simulator reprograms the points  $S_m$  then the verifier of step  $n$  must check a point  $x \in S_m$ . Namely, if we simulate step  $m$  then we do not need  $w_m$ . Therefore the proof must reject: we already argued that all successful provers use all witnesses. But if the verifier of step  $n$  does not query  $x \in S_m$ , then the reprogrammed random oracle will look like the real random oracle to this particular verifier and it must therefore accept the proof (as it accepts the proof when the oracle is reprogrammed, by definition of zero-knowledge).

Let  $x_{m,n}$  denote a query by verifier  $n$  related to proof  $m$ . This is a random variable. The instances  $m$  and  $n$  range from 1 to  $\ell$ , so there are  $\Theta(\ell^2)$  of the random variable  $x_{m,n}$ . The main challenge of the proof is to prove that they are disjoint enough that we force some verifier to make  $\Theta(\ell)$  queries, which is not allowed as we assume the verifier has running time in  $\text{poly}(|\mathcal{R}|, \lambda, \log \ell)$ . The main challenge in proving this is that we cannot make a world where we simulate all proofs, as some verifier will then surely check some reprogrammed point and reject. Also, we cannot easily define  $x_{m,n}$  in the real world where step  $m$  is run honestly, as there is no notion of  $S_m$ . We therefore need to capture  $x_{m,n}$  in the world where step  $m$  is simulated using some poly-time observable. The observable we use is essentially “ $x$  was not queried before step  $m$  and it got queried after step  $m$ ”. We show this captures  $x_{m,n}$  when step  $m$  is simulated. The reason is that by our notion of zero-knowledge a reprogrammed point cannot have been queried before it was reprogrammed. And by arguments from above, some reprogrammed point must be queried by a verifier in the future. We then argue that this poly-time observable  $x_{m,n}$  must exist in the real world too, or zero-knowledge was broken. We then argue that the definitions of the poly-time observables are such that the  $\theta(\ell^2)$  points  $x_{m,n}$  are disjoint enough. This is done in Lemma 4.

*Generalizations.* Our results apply directly to schemes which only rely on random oracles, like that of Valiant [21] (based on CS proofs) and recursive Fractal [9]. However, we emphasise that our results are not oracle separation results. We do not give the adversary access to for instance an NP oracle which can break all cryptography except the random oracle. As a result the impossibility results apply even in presence of additional computational assumptions.

Specifically, the zero-knowledge may depend on computational assumptions, as long as these are not phrased via relativized worlds extra to the random oracle model. The results therefore stand also if there exist for instance trapdoor permutations or indistinguishability obfuscation. Our results do not rule out constructions where zero-knowledge is proven in for instance the generic group model, as it is relativized. Similarly, in result 1 knowledge soundness, and in result 2 the soundness, may depend on computational assumptions, as long as these are not phrased via relativized worlds extra to the random oracle model.

Although we primarily focus on random oracles, the result can easily be generalized to  $O(\text{poly}(\lambda))$ -local oracles, i.e., where responses to queries might be dependent in a bounded manner: All queries can be divided into disjoint sets  $P_i$  of size  $|P_i| = O(\text{poly}(\lambda))$  and replies to queries in different sets are independent. For a given verifier we can simply look at the one which if it queries  $x \in P_i$  then it queries all  $x' \in P_i$ . This still gives it running time  $O(\text{poly}(|\mathcal{R}|, \lambda, \log \ell))$ . And we can now look at the proof system as using a 1-local oracle with larger replies. And it is easy to see that our results still apply to such 1-local oracles. Note that for instance oracles like “generic (bilinear) groups” are *not*  $O(\text{poly}(\lambda))$ -local, as the group law correlates all replies.

## 1.1 Relation to Other Results

The impossibility of Gentry-Wichs [14] for adaptively sound zk-SNARGs applies also to zero-knowledge, non-deterministic IVC, so one cannot hope to construct non-deterministic IVC from falsifiable assumptions. However, this does not rule out a construction of IVC in the RO model. In particular, unlike non-deterministic IVC, there *are* known constructions of zk-SNARKs in the random oracle model, e.g., classic CS proofs [19] from PCPs and the compiler of Ben-Sasson et al. [2] applied to round-by-round sound Holographic IOPs like Fractal [9] and zk-STARKs [1]. Below we compare to other results. The discussion is summarised in Fig. 1.

We note that while Gentry and Wichs [14] proved the impossibility of a security reduction, this paper proves impossibility a construction: Gentry-Wichs shows that any SNARG *cannot have a black-box reduction* to a game-based definition, while ours, shows that any construction of a zero-knowledge non-deterministic IVC in the random oracle model *has an efficient adversary breaking it*.

	CRS	RO	Non-BB RO
IVC ( <b>P</b> )	✓ [17]	?	✓
Batch Arguments ( <b>NP</b> <sup>ℓ</sup> )	✓ [13]	✓	✓
Non-Adaptively Secure SNARGs ( <b>NP</b> )	✓ [18]	✓	✓
Adaptively Secure SNARGs ( <b>NP</b> )	✗ [14]	✓ [19]	✓
Non-Deterministic IVC ( <b>NP</b> )	✗ [14]	✗ [Here]	✓ [21]

**Fig. 1.** An overview of known constructions (✓), impossibility results (✗) and open questions ? in the existing literature, in relation to our result (✗). If a cell has no citation it is implied by the value in another cell in the table, for brevity we only include one construction per cell. Note that Valiant’s original construction [21] of IVC can easily be extended to the non-deterministic setting. CRS stands for the model with a common reference string and no RO. RO stands for the model with a standard RO and no CRS. Non-BB RO stands for the model with non-standard RO *a la* Valiant and no CRS.

*Common Reference String.* A number of recent results have probed the limits of the Gentry and Wichs separation [14] of adaptively secure SNARKs from falsifiable assumptions: Tauman Kalai, Paneth and Yang constructed [17] a delegation scheme for **P**, deterministic IVC, from falsifiable assumptions on bilinear pairings with a CRS. Choudhuri and Jain recently constructed [13] batch arguments for **NP** from standard assumptions and CRS. Lastly Lipmaa and Pavlyk [18] recently resolved an open problem in the Gentry and Wichs paper by proving that there exists a construction of non-adaptively sound SNARGs from falsifiable assumptions.

*Random Oracle.* Adaptively secure (zk)SNARKs has been widely constructed in the random oracle model (without a CRS) [1, 9, 11, 12, 19], including a recent tight lower bound on the number of random oracle queries [16]. We prove that similar positive results cannot be obtained for the *incremental equivalent* of zkSNARK: non-deterministic IVC in the random oracle model. We tackle the impossibility of *non-deterministic* IVC in the random oracle model, since proving the impossibility of *deterministic* IVC (in any model) must preclude the trivial scheme in which the oracle is not used and the poly-log verifier simply decides membership given a poly-log certificate computed by the prover. Impossibility of this seems closely related to proving  $P \not\subseteq \text{NTIME}(O(\log^c n))$ —which remains an open problem in complexity.

*Non-blackbox Random Oracle.* Constructions of (non-deterministic) IVC relying on “non-blackbox” use of the random oracle exists in the literature [9, 21]: in such schemes the security proof is in the random oracle model, however, the construction relies on the oracle having a short description. This is an interesting model where the scheme does not exist in the idealised model in which it is proven secure. Such use of the random oracle often arises implicitly [9, 21] when a SNARK in the RO model is heuristically converted to a SNARK in the plain model, by replacing the random oracle with a concrete cryptographic hash

function, and used to prove the satisfiability of the verification circuit for the same SNARK.

*Non-deterministic IVC in Relativized Worlds.* Non-deterministic IVC trivially exists in worlds with certain types of oracles, the question is how “complicate” this oracle needs to be: motivated both by theoretic curiosity and practical desire to heuristically instantiate the oracle in the standard model. Our results shows that to allow zero-knowledge, incremental PCD the oracle must be non-local.

As discussed above, Chiesa and Liu [8] showed that it is impossible to construct non-trivial PCPs of random oracle computation (e.g., circuits with RO gates). This rules out most hope constructing IVC by proving the correct execution of a verifier in the random oracle model but does not exclude that other design would allow for IVC in the RO model.

On the positive side, the original construction of Proof-Carrying Data (PCD) [10] (a generalization of non-deterministic IVC) by Chiesa and Tromer is in a world with a signed random oracle: a random oracle which additionally returns a signature on the (query, response) pair, this allows verifying the validity of oracle queries without need for oracle computation, by simply verifying the signature, this enables a recursive construction similar to Valiant but without contradictions. This oracle is non-local as all replies are signed with the same key. Recently Chen, Chiesa and Spooner [7] demonstrated that SNARKs exists for the relativized world of low-degree polynomial oracles using an accumulation scheme [6] for oracle query/response pairs. This scheme is non-local as replies are related by the polynomial.

## 1.2 Can We Drop the ZK Assumption?

Our impossibility result applies only to the setting where a large witness is consumed piecemeal and where the proof is zero-knowledge. Since the original construction of Valiant, and modern uses of recursive proofs in the RO model, easily generalises to have these properties the result seems pessimistic, but it keeps open the possibility of getting non-deterministic IVC in the random-oracle model which is *not* zero-knowledge. We prove a secondary result showing that there does not seem to be any easy way to construct this. Namely, the proof system would have to have an unnatural looking property that the proof system itself makes queries it cannot “remember” later. More specifically, we can show that non-deterministic IVC from random oracles is impossible in the following case:

3. If there exists collision intractable hash functions and the proof system has *blackbox knowledge soundness* and the proof system has a property informally stated as follows: it can with non-negligible probability be predicted for all queries made by the *prover* whether they are fresh or it made them before.

This result shows that even if we drop the assumption of zero-knowledge one cannot get incremental proofs, but now using an assumption that the freshness of queries can be determined with non-negligible probability. Note that

this assumption is non-trivial as the proof system is succinct, so it cannot just remember all queries of all previous steps. However, it seems hard to use forgotten queries in a constructive way. We discuss the assumption further in Sect. 5.

For result 3 we use a different proof approach. Here we observe that if the final verifier, of step  $\ell$ , is succinct, then it makes a number of queries to its oracle essentially independent of  $\ell$ . So by setting  $\ell$  large enough we can create a polynomially long *stretch* from step  $p_1$  to step  $p_2$  such that no *fresh* query made by a proof in steps  $[p_1, p_2]$  will be queried by the final verifier. A fresh query is one which was not also made before the stretch. We then create an adversary which picks the witnesses used in steps  $[p_1, p_2]$  independent of the witnesses used outside the interval and independent of all *queries* made before steps  $[p_1, p_2]$ .

During the stretch we let the adversary use a *simulated oracle* instead of the real one for all fresh queries. It simply samples the oracle replies itself without asking the real oracle. This will still give an accepting proof as the final verifier does not make queries corresponding to fresh queries by the prover during the stretch. Hence the real oracle and the simulated one will look the same to the final verifier. Letting the adversary use a simulated oracle  $\tilde{\mathcal{O}}$  during the stretch ensures that the blackbox extractor gets no information on the stretch witnesses: the adversary makes no queries to its oracle during the stretch and is therefore opaque to the blackbox extractor.

Hence all the information that the extractor gets on the stretch witnesses is via queries made by the adversary to its oracle during the proofs *after* step  $p_2$  in the main execution. Intuitively this information can be no larger than the state  $\sigma_2$  of the prover after step  $p_2$ . We could give  $\sigma_2$  to the extractor and let it finish the proof itself. If the proof system is succinct then we can pick  $p_2 - p_1 > |\sigma_2|$  to ensure that  $\sigma_2$  information theoretically cannot encode all the stretch witnesses. This shows that a blackbox extractor cannot compute the stretch witnesses from *blackbox* access to the adversary, violating knowledge soundness.

The above argument uses that we could give the state of the prover after the step  $p_2$  to the adversary and let it finish the proof itself. But note that between steps  $p_1$  and  $p_2$  we used a simulated oracle  $\tilde{\mathcal{O}}$ . To appeal to correctness of the proof system when we let the adversary finish the proof it must know  $\tilde{\mathcal{O}}$  and must be able to determine which queries to send to  $\tilde{\mathcal{O}}$  if they are made again by later steps in the proof. And we should give the adversary this ability by giving it concise information, or we might be leaking the stretch witness to it. We can implement  $\tilde{\mathcal{O}}$  as a pseudo-random oracle and just give the short seed to the adversary. However, we cannot give it the set of all queries made between  $p_1$  and  $p_2$  as the query points themselves might encode information about the stretch witnesses. This is why we need to assume that there is a concise mechanism to determine whether or not a query made by a later step in the proof is fresh, so we know whether to reply with the real random oracle or the simulated  $\tilde{\mathcal{O}}$ . The mechanism need not be perfect. If it passes on a state which is some constant fraction shorter than the stretch witness and works with non-negligible probability we can still get a contradiction to extracting the stretch witnesses when the mechanism works, by making the stretch long enough.

## 2 Definitions

Formally our model of computation is repeated application of a Boolean circuit  $T$  which encodes the “transition function”. Formally, we show impossibility of  $\mathcal{O}$ -IVC supporting particular sets of transition functions  $\mathcal{T}$ , in particular we show impossibility for schemes supporting all Boolean circuits.

**Definition 1 (Transition Functions).** Let  $\mathcal{T}$  be a set of Boolean circuits,  $T \in \mathcal{T}$ :

$$T : \{0, 1\}^{|M|} \times \{0, 1\}^{|w|} \rightarrow \{0, 1\}^{|M|}$$

**Definition 2 (Repeated Application of  $T$ ).** We denote by  $T^\ell$  the function that applies  $T$   $\ell$ -times to a state  $M_0$  with witnesses  $w_1, \dots, w_\ell$ . Formally, let  $T^0 = \text{id}$  (the identity function) and define  $T^\ell$  for  $\ell > 0$  recursively as:

$$\frac{T^\ell(M_0, \vec{w} = (w_1, \dots, w_\ell))}{\begin{array}{l} 1 : M_{\ell-1} \leftarrow T^{\ell-1}(M, (w_1, \dots, w_{\ell-1})) \\ 2 : \text{return } T(M_{\ell-1}, w_\ell) \end{array}}$$

We define the relation/language defined by  $\mathcal{T}$  as follows:

$$(x, \vec{w}) \in \mathcal{R}_{\mathcal{T}} \iff x = (T, M_0, M_\ell, \ell) \wedge M_\ell = T^\ell(M_0, \vec{w})$$

$$x = (T, M_0, M_\ell, \ell) \in \mathcal{L}_{\mathcal{T}} \iff \exists \vec{w} \text{ st. } (x, \vec{w}) \in \mathcal{R}_{\mathcal{T}}$$

**Definition 3 (Non-Deterministic  $\mathcal{O}$ -IVC).** A non-deterministic  $\mathcal{O}$ -IVC scheme for a set of transition functions  $\mathcal{T}$  consists of two PPT  $\mathcal{O}$ -algorithms:

$\mathbb{P}^{\mathcal{O}}(x_\ell = (T, M_0, M_\ell, \ell), w_\ell, \pi_\ell) \mapsto \pi_{\ell+1}$ . A PPT algorithm taking a description of the state transition  $T$ , the initial state  $M_0$ , the current state  $M_\ell$ , the length of the computation  $\ell$ , some additional input  $w_\ell$  and an accepting proof  $\pi_\ell$  of  $x_\ell \in \mathcal{L}_{(T, \ell)}$ . Then outputs a proof  $\pi_{\ell+1}$  of  $x_{\ell+1} = (T, M_0, M_{\ell+1}, \ell + 1) \in \mathcal{L}_{\mathcal{T}}$  where  $M_{\ell+1} = T(M_\ell, w_\ell)$ . Note that the prover is not given the witness for  $x_\ell \in \mathcal{L}_{\mathcal{T}}$ .

$\mathbb{V}^{\mathcal{O}}(x_\ell = (T, M_0, M_\ell, \ell), \pi_\ell) \mapsto \{\top, \perp\}$ . Verifies a proof  $\pi$  of the statement  $(T, M_0, M_\ell, \ell) \in \mathcal{L}_{\mathcal{T}}$ ; i.e., there exists a sequence of witnesses  $\vec{w}$  such that  $M_\ell = T^\ell(M_0, \vec{w})$ .

We assume for notational convenience (and without loss of generality) that the proof for the trivial statement  $x_0 = (T, M_0, M_0, 0)$  (i.e. application of  $T$  zero times to  $M_0$  yields  $M_0$ ) is  $\pi_0 = \epsilon$  (the empty string). Additionally we require that  $\mathbb{P}^{\mathcal{O}}$  and  $\mathbb{V}^{\mathcal{O}}$  satisfy completeness:

**(Perfect) Completeness:** *Informally states that if a proof is produced correctly, it verifies.*

*Formally, for all  $(T, \vec{w}, M_0, \ell)$ :*

$$\Pr \left[ \mathbb{V}^\mathcal{O}(x_\ell = (T, M_0, M_\ell, \ell), \pi_\ell) = \perp \mid \begin{array}{l} \forall i \in [\ell] : \\ M_i = T(M_{i-1}, w_i); \\ x_i = (T, M_0, M_i, i) \\ \pi_i \leftarrow \mathbb{P}^\mathcal{O}(x_i, w_i, \pi_{i-1}) \end{array} \right] = 0$$

*We assume perfect completeness for simplicity, however all our results easily generalize to the slightly weaker case where the scheme has a negligible probability of failure. We do not require that the prover can extend any accepting proof, only those honestly produced.*

*Remark 1.* An alternative definition (similar to [6]) would instead have  $\mathbb{P}^\mathcal{O}$  and  $\mathbb{V}^\mathcal{O}$  take a description of an NP relation  $\mathcal{R}$  rather than a description of a poly-time computable function  $T$ . In which case  $\mathbb{P}$  proves knowledge of a  $w$  st.  $(x = (M, M'), w) \in \mathcal{R}$  (rather than  $M' = T(M, w)$ ). We note that these two definitions are trivially equivalent, but find the definition presented here simpler notationally: in particular the knowledge extractor does not need to explicitly extract a sequence of statements.

We employ both standard soundness and knowledge soundness definitions in different flavors of our impossibility results.

**Definition 4 ((Computationally) Sound Non-Deterministic  $\mathcal{O}$ -IVC).** *The probability of any PPT adversary producing an accepting proof of a false statement is negligible:*

$$\forall \mathcal{A}^{(\cdot)} : \Pr [\mathbb{V}^\mathcal{O}(x, \pi) = \top \wedge x \notin \mathcal{L} \mid (x, \pi) \leftarrow \mathcal{A}^\mathcal{O}(1^\lambda);] \leq \text{negl}(\lambda)$$

Many languages are trivial (i.e., every instance is in the language), in which case knowledge soundness is required for non-deterministic IVC to be non-trivial. We consider two standard variations: (1) knowledge soundness with a universal non-blackbox extractor (the weaker definition), in which the extractor is given access to the code of the adversary. (2) knowledge soundness with a blackbox extractor (the stronger definition), in which the extractor is given only blackbox (rewinding) access to the adversary.

**Definition 5 (Universal Non-Blackbox Knowledge Soundness Non-Deterministic  $\mathcal{O}$ -IVC).** *There exists a PPT algorithm  $\mathbb{E}$  st. for all PPT  $\mathcal{A}^\mathcal{O}$  when  $\mathcal{A}^\mathcal{O}$  outputs an accepting proof, the extractor given a description of the adversary, recovers a valid witness  $(w_1, \dots, w_\ell)$  given  $\mathcal{A}^{(\cdot)}$  except with negligible probability. Formally:*

$$\exists \mathbb{E} \text{ st. } \forall \mathcal{A}^{(\cdot)} : \Pr \left[ \begin{array}{l} \mathbb{V}^\mathcal{O}(x, \pi) = \top \\ \wedge T^\ell(M_0, \vec{w}) \neq M_\ell \end{array} \mid \begin{array}{l} (x, \pi) \leftarrow \mathcal{A}^\mathcal{O}(1^\lambda); \vec{w} \leftarrow \mathbb{E}^\mathcal{O}(1^\lambda, x, \mathcal{A}^{(\cdot)}); \\ x = (T, M_0, M_\ell, \ell) \end{array} \right] \leq \text{negl}(\lambda)$$

**Definition 6 (Blackbox Knowledge Sound Non-Deterministic  $\mathcal{O}$ -IVC).**

There exists a PPT algorithm  $\mathbb{E}$  st. for all PPT  $\mathcal{A}^{\mathcal{O}}$  when  $\mathcal{A}^{\mathcal{O}}$  outputs an accepting proof, the extractor given black-box (rewinding) access to the adversary  $\mathcal{A}^{(\cdot)}$  recovers a valid witness  $(w_1, \dots, w_\ell)$ , except with negligible probability. Formally:

$$\Pr \left[ \begin{array}{l} \mathbb{V}^{\mathcal{O}}(x, \pi) = \top \\ \wedge T^\ell(M_0, \vec{w}) \neq M_\ell \end{array} \middle| \begin{array}{l} (x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda); \vec{w} \leftarrow \mathbb{E}^{\mathcal{O}, \mathcal{A}^{(\cdot)}}(1^\lambda, x); \\ x = (T, M_0, M_\ell, \ell) \end{array} \right] \leq \text{negl}(\lambda)$$

Additionally we may require that the IVC scheme is zero-knowledge, which informally states that any step can be simulated by programming the oracle and that simulated proofs are indistinguishable from real proofs. Note that the statement to be simulated includes an accepting proof of correctness for  $M_\ell$ .

**Definition 7 ((Computational) Zero-Knowledge Non-Deterministic  $\mathcal{O}$ -IVC).**

There exists a PPT (in  $\lambda, |T|, \ell$ ) algorithm  $\mathbb{S}^{(\cdot)}$  which for any  $T \in \mathcal{T}$ ,  $\ell = \text{poly}(\lambda)$ ,  $x = (T, M_0, M_\ell, \ell) \in \mathcal{L}_T$ ,  $w$ , and accepting  $\pi$  ( $\mathbb{V}^{\mathcal{O}}(x, \pi) = \top$ ),  $\mathbb{S}^{\mathcal{O}}$  outputs an accepting proof and a set of (re)programmings  $\mathcal{Q} = \{(Q_i, R_i)\}_i$  for the oracle fooling any PPT adversary.

$$\exists \mathbb{S}^{\mathcal{O}} \forall \mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \forall T \in \mathcal{T}, x = (T, M_0, M_\ell, \ell) \in \mathcal{L}_T, w, \pi \text{ st. } \mathbb{V}^{\mathcal{O}}(x, \pi) = \top :$$

$$\Pr \left[ b = b' \middle| \begin{array}{l} M_{\ell+1} = T(M_\ell, w) \\ \mathbf{h} \leftarrow \mathcal{A}_1^{\mathcal{O}}(1^\lambda, M_{\ell+1}, x, \pi) \\ \pi'_0 \leftarrow \mathbb{P}^{\mathcal{O}}(x, \pi, w) \\ (\mathcal{Q}, \pi'_1) \leftarrow \mathbb{S}^{\mathcal{O}}(M_{\ell+1}, x, \pi) \\ b \leftarrow_{\mathbb{S}} \{0, 1\}; \\ b' \leftarrow \mathcal{A}_2^{\mathcal{O}b}(1^\lambda, \mathbf{h}, \pi'_b) \end{array} \right] - 1/2 \leq \text{negl}(\lambda)$$

Where  $\mathcal{O}_0 = \mathcal{O}$  and  $\mathcal{O}_1 = [\mathcal{Q}, \mathcal{O}]$ , where  $[\mathcal{Q}, \mathcal{O}]$  is the oracle mapping  $\mathfrak{q}$  to  $R_i$  if  $(\mathfrak{q}, R_i) \in \mathcal{Q}$  and  $\mathcal{O}(\mathfrak{q})$  otherwise. The probability is over  $\mathcal{O}$ , the random tape of  $\mathcal{A}^{(\cdot)}$ ,  $\mathbb{P}$  and  $\mathbb{S}$ . We allow the running time of the simulator to depend polynomially on the running time of the adversary.

*Remark 2.* An easy observation is that if  $\mathcal{A}_1$  queried  $\mathcal{O}$  at  $\mathfrak{q}$  in  $\mathbf{h} \leftarrow \mathcal{A}_1^{\mathcal{O}}(1^\lambda, M_{\ell+1}, x, \pi)$ , then we can assume that, except with negligible probability,  $\mathcal{O}_1(\mathfrak{q}) = \mathcal{O}(\mathfrak{q})$ , i.e., the simulator does not reprogram on  $\mathfrak{q}$ . Namely, if  $\mathcal{O}_1(\mathfrak{q}) \neq \mathcal{O}(\mathfrak{q})$  happens with non-negligible probability the adversary could remember all queries  $\mathfrak{q}$  and replies made during the first step and redo them in the second step and guess  $b = 1$  when  $\mathcal{O}_1(\mathfrak{q}) \neq \mathcal{O}(\mathfrak{q})$  and  $b = 0$  otherwise. This would break zero-knowledge. We call the property that the simulator only programs points that were never queried *fresh reprogramming* below.



*Remark 3.* We want to warn that our definitions were tailored for proving negative results. They might not be strong enough for positive applications. Namely, our soundness requires only that extension works for honestly generated proofs. So it might be possible to maliciously generate a proof  $\pi_{i-1}$  which accepts but extends into a non-accepting proof  $\pi_i$ . That means that in a proof carrying data context an honest party might end up producing and further extending a non-accepting proof  $\pi_i$  into a proof  $\pi_{i+1}$ . At the same time by our notion of zero-knowledge the proof  $\pi_{i+1}$  might not be zero-knowledge as zero-knowledge only holds when starting from an accepting proof  $\pi = \pi_i$ . That means an honest party might end up producing a non-zero-knowledge proof  $\pi_{i+1}$ . However, the definitions are enough to prove our results. Starting from a weaker definition makes impossibility proofs stronger. For practical applications stronger definitions should be used.

### 2.1 Rerandomizable Commitments

**Definition 8 (Rerandomizable Bit Commitments).** *A rerandomizable bit commitment scheme consists of three algorithms:*

**Setup** :  $\{1\}^* \times \{0, 1\}^* \rightarrow \mathcal{P}$  a PPT algorithm which takes a unary representation of the security parameter  $1^\lambda$  and produces public parameters, i.e.,  $\text{pp} \leftarrow \text{Setup}(1^\lambda; r)$  for a random tape  $r \in \{0, 1\}^*$ .

**Commit** :  $\mathcal{P} \times \{0, 1\} \rightarrow \mathcal{C}$  a deterministic algorithm which sends a bit to the commitment space, i.e.,  $\mathbf{c} = \text{Commit}(\text{pp}, b)$ ,  $b \in \{0, 1\}$ .

**ReRand** :  $\mathcal{P} \times \mathcal{C} \times (\{0, 1\}^*)^m \rightarrow \mathcal{C}$  takes a commitment and produces a rerandomization of the same commitment (without knowing the opening).

Note that we do not require the rerandomizable commitments to have succinct openings, in particular “Open” can be constructed by simply re-executing all the rerandomizations of the original commitment, i.e.  $\text{Open}(\text{pp}, b, \mathbf{c}, \mathbf{r} = (r_1, \dots, r_m)) := \mathbf{c} \stackrel{?}{=} \text{ReRand}^m(\text{pp}, \text{Commit}(\text{pp}, b); \mathbf{r}) = \text{ReRand}(\dots \text{ReRand}(\text{ReRand}(\text{pp}, \text{Commit}(\text{pp}, b); r_1); r_2), \dots; r_m)$

$\text{Game}_{\text{Hiding}}^{(m)}(\mathcal{A}, \lambda)$
1 : $\text{pp} \leftarrow \text{Setup}(1^\lambda)$
2 : $((v^{(0)}, \tilde{r}^{(0)}), (v^{(1)}, \tilde{r}^{(1)}), \text{st}) \leftarrow \mathcal{A}(\text{find}, \text{pp}, 1^\lambda)$
3 : $\mathbf{c}^{(0)} = \text{ReRand}^m(\text{Commit}(\text{pp}, v^{(0)}); \tilde{r}^{(0)})$
4 : $\mathbf{c}^{(1)} = \text{ReRand}^m(\text{Commit}(\text{pp}, v^{(1)}); \tilde{r}^{(1)})$
5 : $b \leftarrow_{\$} \{0, 1\}; \mathbf{c}' \leftarrow \text{ReRand}(\mathbf{c}^{(b)})$
6 : $b' \leftarrow \mathcal{A}(\text{guess}, \text{st}, \text{pp}, \mathbf{c}', 1^\lambda)$
7 : <b>return</b> $b \stackrel{?}{=} b'$

We require the rerandomizable commitment scheme to be perfectly binding and computationally hiding.

**Definition 9 (Perfect Binding).** For every  $\text{pp}$  and number of rerandomizations  $m$ , the set of (rerandomized) commitments to 0 and 1 are disjoint, i.e.

$$\forall m \geq 0, \forall \mathbf{r}^{(0)}, \mathbf{r}^{(1)} : \Pr \left[ \mathbf{c}_0 = \mathbf{c}_1 \mid \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda) \\ \mathbf{c}_0 = \text{ReRand}^m(\text{pp}, \text{Commit}(\text{pp}, 0), \mathbf{r}^{(0)}) \\ \mathbf{c}_1 = \text{ReRand}^m(\text{pp}, \text{Commit}(\text{pp}, 1), \mathbf{r}^{(1)}) \end{array} \right] = 0$$

We do not require this to hold if the two commitments are rerandomized a different number of times; which is weaker than the common definition.

**Definition 10 (Computational Hiding).** For every  $m \geq 1$  and PPT adversary  $\mathcal{A}$ , there exists a negligible function  $\text{negl}(\lambda)$  such that:

$$\Pr \left[ \text{Game}_{\text{Hiding}}^{(m)}(\mathcal{A}, \lambda) \right] - 1/2 \leq \text{negl}(\lambda)$$

We do not require the scheme to hide the number ( $m$ ) of times a commitment has been rerandomized; which is weaker than the common definition.

*Remark 4 (Concrete Assumptions for Perfectly Binding Rerandomizable Commitments).* Perfectly binding rerandomizable commitments can be obtained from decisional Diffie-Hellman using Elgamal encryption: which additionally hides the number ( $m$ ) of rerandomizations; a property we do not require.

## 2.2 Collision Intractable Hashes

**Definition 11 (Collision Intractable Hash Functions).** A family  $\mathcal{H}_\lambda = \{\text{H}_k\}_{k \in \{0,1\}^\lambda}$  of a set of PPT computable functions from  $\{0,1\}^*$  to  $\{0,1\}^\lambda$  indexed by the security  $\lambda$  is collision intractable if for every PPT adversary  $\mathcal{A}$ , there exist a negligible function  $\text{negl}(\lambda)$  st.

$$\Pr \left[ \text{H}(x) = \text{H}(x') \wedge x \neq x' \mid \text{H} \leftarrow \mathcal{H}_\lambda; (x, x') \leftarrow \mathcal{A}(\text{H}, 1^\lambda) \right] \leq \text{negl}(\lambda) .$$

## 2.3 Basic Notation

**Definition 12 (Stretch).** Let a length  $\ell$  of a proof be fixed, i.e.,  $\ell$  is the number of times the basic step function is run. We call  $(p, q)$  with  $1 \leq p$ ,  $0 \leq q$  and  $p + q \leq \ell$  a stretch of length  $q$  with start position  $p$ .

**Definition 13 (Query Sets).** Consider a length  $\ell$  and a run of a proof of length  $\ell$ , which proceeds as follows. For  $i = 1, \dots, \ell$  compute  $M_i = T(M_{i-1}, w_i)$ , let  $\mathcal{P}_\downarrow^{(i)}$  be the queries made to  $\mathcal{O}$  in computing  $\pi_i = \mathbb{P}^\mathcal{O}(T, M_{i-1}, \pi_{i-1}, w_i; \rho_i)$  and let  $\mathcal{V}_\downarrow^{(i)}$

be the queries made to  $\mathcal{O}$  in computing  $\mathbb{V}^{\mathcal{O}}(T, M_0, M_i, \pi_i)$ . For  $1 \leq i \leq k \leq \ell$ , let  $\mathcal{V}_{\cup}^{(i,k)} = \cup_{j=i}^k \mathcal{V}_{\downarrow}^{(j)}$  and  $\mathcal{P}_{\cup}^{(i,k)} = \cup_{j=i}^k \mathcal{P}_{\downarrow}^{(j)}$ . Define the ‘fresh’ queries made at step  $i$  as  $\mathcal{V}_{\Delta}^{(i)} = \mathcal{V}_{\downarrow}^{(i)} \setminus \mathcal{V}_{\cup}^{(1,i-1)}$  and  $\mathcal{P}_{\Delta}^{(i)} = \mathcal{P}_{\downarrow}^{(i)} \setminus \mathcal{P}_{\cup}^{(1,i-1)}$ . Finally define the fresh queries during stretches as  $\mathcal{V}_{\Delta}^{(p,q)} = \cup_{i=p}^{p+q-1} \mathcal{V}_{\Delta}^{(i)}$  and  $\mathcal{P}_{\Delta}^{(p,q)} = \cup_{i=p}^{p+q-1} \mathcal{P}_{\Delta}^{(i)}$ .

**Definition 14 (Oracle Extension).** For a set of queries  $\mathcal{Q}_1$  and two oracles  $\mathcal{O}_1$  and  $\mathcal{O}$  we define the oracle  $[\mathcal{Q}_1 \mapsto \mathcal{O}_1, \mathcal{O}]$  as follows. On input  $\mathfrak{q}$ , if  $\mathfrak{q} \in \mathcal{Q}_1$  then output  $\mathcal{O}_1(\mathfrak{q})$ . Otherwise output  $\mathcal{O}(\mathfrak{q})$ . In general, let

$$[\mathcal{Q}_1 \mapsto \mathcal{O}_1, \dots, \mathcal{Q}_{\ell} \mapsto \mathcal{O}_{\ell}, \mathcal{O}] = [\mathcal{Q}_1 \mapsto \mathcal{O}_1, [\mathcal{Q}_2 \mapsto \mathcal{O}_2, \dots, \mathcal{Q}_{\ell} \mapsto \mathcal{O}_{\ell}, \mathcal{O}]].$$

### 3 Theorem Statements

Having the definitions in place we give the formal theorem statements. For the statements we use the following step functions. Let  $T_H$  be the step function for repeated hashing of the witnesses, i.e.,  $T_H := H(M||w)$ . Let  $T_{pp}$  be the step function for repeated rerandomization of a commitment using the witness as randomness, i.e.,  $T_{pp}(M, w) := \text{ReRand}(pp, M; w)$ . The following statements is proven in Sect. 4.

**Theorem 1 (Impossibility of Non-Trivial ZK Non-Deterministic  $\mathcal{O}$ -IVC).** *The existence of collision intractable functions or perfectly binding rerandomizable commitments precludes the existence of (knowledge-sound) non-trivial zero-knowledge non-deterministic  $\mathcal{O}$ -IVC, more formally:*

- **Collision Intractability Precludes Knowledge-Soundness.** *Assuming the existence of a family of collision intractable functions  $\mathcal{H}_{\lambda}$  (Definition 11), there exists a transition function  $T_H$  such that any zero-knowledge (Definition 7), knowledge-sound (Definition 5)  $\mathcal{O}$ -IVC scheme (Definition 3) for the step function  $T_H$  must have a verifier with running time linear in the number of steps  $\ell$ .*
- **Rerandomizable Commitments Precludes (Regular) Soundness.** *Assuming the existence of perfectly binding rerandomizable commitment schemes (Definition 8), there exists transition functions  $T_{pp}$  such that any zero-knowledge (Definition 7) and computationally sound (Definition 4)  $\mathcal{O}$ -IVC scheme (Definition 3) for  $T_{pp}$  must have a verifier with running time linear in the number of steps  $\ell$ .*

For the impossibility for black-box schemes we need to formalize the notion that one can recognize whether queries are fresh.

**Definition 15 (Structured Oracle Queries).** *We say that a proof system  $(\mathcal{T}, \mathbb{P}, \mathbb{V})$  has structured oracle queries if there exists a PPT algorithm used for which the following holds for all PPT adversaries  $\mathcal{A}$ . For all  $T \in \mathcal{T}$ , all lengths  $\ell$ , and all witnesses  $(w_1, \dots, w_{\ell})$  let  $M_0$  be an initial state,  $\pi_0 = \epsilon$ ,  $\pi_i = \mathbb{P}^{\mathcal{O}}(T, M_{i-1}, \pi_{i-1}, w_i, \rho_i)$ , where  $\rho_i$  is the possible random tape of  $\mathbb{P}$ ,  $\mathcal{P}_{\downarrow}^{(i)}$*

be the queries made by this  $i$ 'th run of  $\mathbb{P}$ ,  $\mathcal{P}_{\cup}^{(1,i)} = \cup_{j=1}^i \mathcal{P}_{\downarrow}^{(j)}$ , and let  $\text{used}_i = \text{used}(T, M_{i-1}, \pi_{i-1}, w_i, \rho_i)$  be the description of a PPT predicate. Now compute  $(i, \mathfrak{q}) = \mathcal{A}^{\mathcal{O}}(T, \vec{w}, M_0, \vec{\rho})$ . We say that the adversary wins if  $\text{used}_i(\mathfrak{q}) = \top$  and  $\mathfrak{q} \notin \mathcal{P}_{\cup}^{(1,i)}$  or  $\text{used}_i(\mathfrak{q}) = \perp$  and  $\mathfrak{q} \in \mathcal{P}_{\cup}^{(1,i)}$ . We say that the proof system is  $p_{\text{STRUC-SOQ}}$  if the probability that the adversary wins is  $\leq 1 - p_{\text{STRUC}}$ .

Below we will assume that the proof system is  $1/\lambda^\gamma$ -SOQ for some constant  $\gamma > 0$ . This means we essentially just need a non-negligible probability that the queries are structured. Note that  $\text{used}_i$  is computed from the current state of the prover, so if the proof system is succinct then so is the state needed to compute  $\text{used}_i$  which will be basis for our impossibility result. The following theorem is proven in Sect. 5.

**Theorem 2.** *If there exist collision intractable hash functions then there does not exist succinct, non-deterministic IVC for the random oracle model (Definition 3) with blackbox knowledge soundness (Definition 6) which is  $1/\lambda^{\mathcal{O}(1)}$ -SOQ (Definition 15) for the step function  $T_{\text{H}}$ . By succinct we mean that the size of a proof of an  $\ell$ -iteration computation is  $\text{poly}(\lambda, \log \ell)$ .*

### 4 Impossibility from Zero-Knowledge

In the following section we prove two impossibility results for the case where the  $\mathcal{O}$ -IVC is zero-knowledge. One is for the case where the proof system is knowledge sound and collision intractable functions exists. The other is for the case where the proof system has just soundness but under the assumption of perfectly binding rerandomizable commitments. We start by proving some lemmas and then put them together at the end of the section.

The following lemmas state that for certain transition functions no adversary can produce an accepting proof without knowing the witness for every step; without violating (knowledge) soundness of the  $\mathcal{O}$ -IVC scheme.

Let  $\mathcal{U}_n^\ell = \mathcal{U}_n \times \dots \times \mathcal{U}_n$  be the distribution of  $\ell$  iid. uniform  $n$  bit strings and define  $\vec{w}^{(\bar{m})} := (w_1, \dots, w_{m-1}, \perp, w_{m+1}, \dots, w_\ell)$  (i.e., a sequence where the  $m$ 'th witness is removed) for any sequence of witnesses  $\vec{w}$ . Impossibility of knowledge soundness follows from collision intractable functions:

**Lemma 1 (All Witnesses are Required for Knowledge Soundness).** *For a (randomly sampled) collision intractable hash function  $\text{H} : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ , consider the following step function  $T_{\text{H}}(M, w) := \text{H}(M||w)$ . We now show that for any knowledge-sound  $\mathcal{O}$ -IVC scheme, PPT adversary  $\hat{\mathcal{A}}$ ,  $\ell = \mathcal{O}(\text{poly}(\lambda, |T_{\text{H}}|))$  and  $m \in [\ell]$ ,  $\hat{\mathcal{A}}$  produces an accepting proof  $\pi$  of the  $T_{\text{H}}^\ell$  execution given all witnesses except for step  $m$ , with only negligible probability. i.e., there exists a negligible function  $\text{negl}(\lambda)$  st.*

$$\Pr \left[ \mathbb{V}^{\mathcal{O}}(x, \pi) = \top \mid \begin{array}{l} \text{H} \leftarrow_{\$} \mathcal{H}_\lambda; \vec{w} \leftarrow_{\$} \mathcal{U}_{2\lambda}^\ell; \\ M_0 = \epsilon; \text{ for } i \in [\ell] : M_i = T_{\text{H}}(M_{m-1}, w_m); \\ \vec{w}^{(\bar{m})} = (w_1, \dots, w_{m-1}, \perp, w_{m+1}, \dots, w_\ell); \\ \pi \leftarrow \hat{\mathcal{A}}^{\mathcal{O}}(x = (T_{\text{H}}, M_0, M_\ell, \ell), \vec{w}^{(\bar{m})}, M_m) \end{array} \right] \leq \text{negl}(\lambda)$$

*Proof.* Since the  $\mathcal{O}$ -IVC scheme is (non-blackbox) extractable by assumption, there exists an extractor  $\mathbb{E}$ . Now, for any  $\ell \geq 1$  and  $m \in [\ell]$ , consider the following adversary  $\mathcal{A}$  for the collision game (see Definition 11):

```

(v1, v2) ← A(H)
// Run A-hat to get a proof without the pre-image of Mm
1 : w-vec ← sU_{2^lambda}^ell
2 : M0 = epsilon; for i in [ell] : Mi = TH(Mi-1, wi);
3 : w-vec^(m-hat) := (w1, ..., w_{m-1}, perp, w_{m+1}, ..., w_ell)
4 : x = (TH, epsilon, M_ell, ell); pi ← A-hat^O(x, w-vec^(m-hat), Mm)
// Run E to get preimages for each state.
5 : w-prime ← E(x, A-hat^{(cdot)}(x, w-vec^(m-hat), Mm))
6 : M'0 = epsilon; for i in [ell] : M'i = TH(M'i-1, w'i);
// Look for collision.
7 : for i in [ell - 1] :
8 :     v1 := Mi || wi+1; v2 := M'i || w'i+1
9 :     if Mi+1 = M'i+1 ^ v1 != v2
10 :         return (v1, v2)
11 : return perp
    
```

Let  $f : \{0, 1\}^{2\lambda} \rightarrow \{0, 1\}^\lambda$  be defined as  $f(y) \mapsto H(M_{m-1} || y)$ , note that the probability that there exists  $\geq 2$  preimages of  $f(w_m)$  is overwhelming, since  $w_m$  is sampled uniformly at random. Hence the extractor given only  $M_m := f(w_m)$  recovers  $w'_m$  such that  $w_m \neq w'_m$  with probability at least  $1/2 - \text{negl}(\lambda)$ . This violates collision intractability of  $f$  and in particular of  $H$ .  $\square$

If we are willing to make the stronger assumption that perfectly binding and computationally hiding rerandomizable commitments exist we can strengthen the lemma to violate soundness of the  $\mathcal{O}$ -IVC scheme:

**Lemma 2 (All Witnesses are Required for Soundness).** *For a perfectly binding rerandomizable commitment scheme, consider the following step function  $T_{pp}(M, w) := \text{ReRand}(pp, M; w)$ —repeated rerandomization of the commitment. We now show that for any computationally sound  $\mathcal{O}$ -IVC scheme, PPT adversary  $\hat{\mathcal{A}}, \ell = O(\text{poly}(\lambda, |T_{pp}|))$  and  $m \in [\ell]$ ,  $\hat{\mathcal{A}}$  produces an accepting proof  $\pi$  of the  $T_{pp}^\ell$  execution given all witnesses except for step  $m$ , with only negligible probability, i.e., there exists a negligible function  $\text{negl}(\lambda)$  st.*

$$\Pr \left[ \mathbb{V}^{\mathcal{O}}(x, \pi) = \top \mid \begin{array}{l} \text{pp} \leftarrow_{\$} \text{Setup}(1^\lambda); \vec{w} \leftarrow_{\$} \mathcal{U}_{\text{poly}}^\ell; \\ M_0 = \text{Commit}(\text{pp}, 0); \\ \text{for } i \in [\ell] : M_i = T_{\text{pp}}(M_{m-1}, w_m); \\ \vec{w}^{(\bar{m})} = (w_1, \dots, w_{m-1}, \perp, w_{m+1}, \dots, w_\ell); \\ \pi \leftarrow \hat{\mathcal{A}}^{\mathcal{O}}(x = (T_{\text{pp}}, M_0, M_\ell, \ell), \vec{w}^{(\bar{m})}, M_m) \end{array} \right] \leq \text{negl}(\lambda)$$

*Proof.* Let  $p$  be the probability that  $\hat{\mathcal{A}}$  outputs an accepting proof (in the original game), we assume for contradiction that  $p$  is non-negligible (in  $\lambda$ ). Consider the following PPT algorithm which we use to violate soundness of the  $\mathcal{O}$ -IVC scheme or break computational hiding of the commitment scheme:

$\mathcal{A}_{\text{Hiding}}(\text{find}, \text{pp}, 1^\lambda)$	$\mathcal{A}_{\text{Hiding}}(\text{guess}, \text{st}, \text{pp}, ', 1^\lambda)$
// Sample randomness / witnesses for $\ell$ steps. 1 : $\vec{r} \leftarrow_{\$} \mathcal{U}_{\text{poly}}^\ell; \text{st} = \vec{r}$ // Get rerandomisation of either 0 or 1. 2 : $v^{(0)} = 0; v^{(1)} = 1$ 3 : $\vec{r}^{(0)} = \vec{r}^{(1)} = (r_1, \dots, r_{m-1})$ 4 : <b>return</b> $((v^{(0)}, \vec{r}^{(0)}), (v^{(1)}, \vec{r}^{(1)}), \text{st})$	1 : $\vec{r} = \text{st}$ 2 : $\vec{w}^{(\bar{m})} := (r_1, \dots, r_{m-1}, \perp, r_{m+1}, \dots, r_\ell)$ 3 : $M_0 = \text{Commit}(\text{pp}, 0); M_m = '$ 4 : <b>for</b> $i \in [m+1, \ell] : M_i = T_{\text{pp}}$ $(M_{i-1}, w_i)$ 5 : $x = (T_{\text{pp}}, M_0, M_\ell, \ell)$ 6 : $\pi \leftarrow \hat{\mathcal{A}}^{\mathcal{O}}(x, \vec{w}^{(\bar{m})}, M_m)$ 7 : <b>if</b> $\mathbb{V}^{\mathcal{O}}(x, \pi) = 1$ <b>return</b> 0 8 : <b>else return</b> 1

Observe that when  $b = 0$  in the  $\text{Game}_{\text{Hiding}}^{(m-1)}$  game  $M_m = c'$  is a rerandomization of the 0 commitment and hence  $M_\ell$  is as well, therefore  $x$  is true and  $\mathcal{A}_{\text{Hiding}}$  correctly returns 0 with probability  $p$  by assumption on  $\hat{\mathcal{A}}$ . However, when  $b = 1$ , the commitment  $c'$  is a rerandomization of the 1 commitment and  $x$  is false, hence  $\mathbb{V}^{\mathcal{O}}(x, \pi) = 0$  except with negligible probability  $\text{negl}(\lambda)$ , otherwise computational soundness is violated. This implies that  $\mathcal{A}_{\text{Hiding}}$  wins the  $\text{Game}_{\text{Hiding}}^{(m)}$  game with advantage at least  $p - \text{negl}(\lambda)/2$ ; which is non-negligible, a contradiction.  $\square$

We now show that proof systems with transition functions like the ones in Lemma 1 and Lemma 2 where all witnesses are needed will have the verifiers make many queries to the random oracle.

In the below we use some common definitions of honest and simulated experiments. For a given proof system we can define the honest experiment  $\text{HonExp}$  as follows.

**Experiment HonExp:**

1. Let  $M_0$  be the start state and  $\pi_0 = \epsilon$ . Let  $\vec{w}$  be a vector of witnesses.
2. For  $i = 1, \dots, \ell$  compute  $M_i = T(M_{i-1}, w_i)$ ,  $\pi_i = \mathbb{P}^{\mathcal{O}}(T, M_{i-1}, \pi_{i-1}, w_i)$ ,  $\mathbb{V}^{\mathcal{O}}(T, M_0, M_i, \pi_i)$ .

Let the query sets be defined as in Definition 13.

For any  $1 \leq m \leq \ell$  we can define a simulation experiment  $\text{SimExp}_m$  where everything is defined as in the honest experiment except that we simulate in step  $m$  and then use the reprogrammed oracle from then on.

**Experiment SimExp<sub>m</sub>:**

1. Let  $M_0$  be the start state and  $\pi_0 = \epsilon$ . Let  $\vec{w}$  be a vector of witnesses.
2. For  $i = 1, \dots, m - 1$  compute  $M_i = T(M_{i-1}, w_i)$ ,  $\pi_i = \mathbb{P}^{\mathcal{O}}(T, M_{i-1}, \pi_{i-1}, w_i)$ ,  $\mathbb{V}^{\mathcal{O}}(T, M_0, M_i, \pi_i)$ .
3. Compute  $M_m = T(M_{m-1}, w_m)$ . Compute a simulated proof  $(\mathcal{Q}, \pi_m) \leftarrow \mathbb{S}^{\mathcal{O}}(T, M_m, \pi_{m-1})$ . Let  $\mathcal{O}_1 = [\mathcal{Q}, \mathcal{O}]$ . Let  $S_m = \{\mathfrak{q} \mid \exists y ((\mathfrak{q}, y) \in \mathcal{Q})\}$  be the set of query points on which  $\mathcal{Q}$  programs. Compute  $\mathbb{V}^{\mathcal{O}_1}(T, M_0, M_m, \pi_m)$ . Note that we use the reprogrammed oracle from here on.
4. For  $i = 1, \dots, m + 1$  compute  $M_i = T(M_{i-1}, w_i)$ ,  $\pi_i = \mathbb{P}^{\mathcal{O}_1}(T, M_{i-1}, \pi_{i-1}, w_i)$ ,  $\mathbb{V}^{\mathcal{O}_1}(T, M_0, M_i, \pi_i)$ .

We can show that if all steps are run honestly except that step  $m$  is simulated then *all* future verifiers must check one of the points that the simulator programmed in step  $m$ . More formally:

**Lemma 3 (Must Check Programmed Points).** *For transition functions  $T$  as described in Lemma 1 and Lemma 2 and for all  $m$  and  $\ell$  with  $1 \leq m \leq \ell$  it holds that  $\Pr[S_m \cap \mathcal{V}_\downarrow^{(\ell)} = \emptyset] = \text{negl}(\lambda)$  for a negligible function  $\text{negl}(\lambda)$ .*

*Proof.* Towards contradiction, suppose there exists  $(m, \ell)$  such that  $\Pr[S_m \cap \mathcal{V}_\downarrow^{(\ell)} = \emptyset] = p$  where  $p$  is non-negligible in  $\lambda$ , then construct an adversary violating Lemma 1 and Lemma 2 as follows:

Where  $T, M_0$  are instantiated as in Lemma 1 and Lemma 2. To reach contradiction we now argue that  $\mathbb{V}^{\mathcal{O}}(x, \pi) = \top$  with probability  $p$ : notice that when  $S_m \cap \mathcal{V}_\downarrow^{(\ell)} = \emptyset$ , then  $\mathbb{V}^{[\mathcal{Q}_m, \mathcal{O}]}(x, \pi) = \mathbb{V}^{\mathcal{O}}(x, \pi)$  since the verifier makes no queries in  $\mathcal{Q}_m(S_m)$ . Now, simply observe that  $\mathbb{V}^{[\mathcal{Q}_m, \mathcal{O}]}(x, \pi) = \top$  follows from zero-knowledge—otherwise  $\pi_m$  could be distinguished from a real proof by extending it  $\ell - m - 1$  times and running the verifier. Therefore  $\mathbb{V}^{\mathcal{O}}(x, \pi)$  accepts with non-negligible probability contradicting Lemma 1 and Lemma 2.  $\square$

The above lemma intuitively implies that some query points “belonging” to step  $m$  must be checked by many future verifiers. If this was true for all  $m$

```

 $\pi \leftarrow \hat{\mathcal{A}}^{\mathcal{O}}(x, \vec{w}^{(\bar{m})}, M_m)$ ; produces a proof without  $w_m$ .
1 :  $x = (T, M_0, M_\ell, \ell)$ ;  $\pi_0 = \epsilon$ 
2 :  $\vec{w}^{(\bar{m})} = (w_1, \dots, w_{m-1}, \perp, w_{m+1}, \dots, w_\ell)$ 
   // Start execution using first  $m - 1$  witnesses
3 : for  $j \in [1, m - 1]$  :
4 :    $M_j \leftarrow T(M_{j-1}, w_j)$ 
5 :    $\pi_j \leftarrow \mathbb{P}^{\mathcal{O}}(T, M_{j-1}, w_j, \pi_{j-1})$ 
   // Simulate step  $m$ 
6 :    $(\mathcal{Q}_m, \pi_m) \leftarrow \mathbb{S}^{\mathcal{O}}(T, M_m, \pi_{m-1})$ 
   // Finish execution with reprogrammed oracle
7 :   for  $j \in [m + 1, \ell]$  :
8 :      $M_j \leftarrow T(M_{j-1}, w_j)$ 
9 :      $\pi_j \leftarrow \mathbb{P}^{[\mathcal{Q}_m, \mathcal{O}]}(T, M_{j-1}, w_j, \pi_{j-1})$ 
10 : return  $\pi_\ell$ 

```

simultaneously and these query points were distinct then we would be done. Too many distinct points would need to be checked often in the future, so the query sets of the verifiers would have to get too big. It is, however, not straight forward to generalise the above lemma to show that the query sets of the verifiers must be large. If we simulate at many steps the set of reprogrammed points might grow so large that we cannot argue that the final verifier will not query a reprogrammed point and reject the proof. Note that the final verifier has access to the real random oracle, not the reprogrammed random oracle. And if the verifier rejects, then we do not get a contradiction to Lemma 1 or Lemma 2. We will therefore need a slightly more subtle strategy. We show that because Lemma 3 holds in  $\text{SimExp}_m$  we can carefully compute in  $\text{HonExp}$  a set of query points uniquely associated to step  $m$  which must be checked often in the future. In  $\text{HonExp}$  we can then sum over all  $m$ .

**Lemma 4 (Too Large Verifier Query Set).** *For transition functions  $T$  for which the property in Lemma 3 holds there exists  $i$  such that the set  $\mathcal{V}_\downarrow^{(i)}$  sometimes has size at least  $\frac{\ell-1}{4}$  in  $\text{HonExp}$ .*

*Proof.* We describe an adversary  $\mathcal{B}$  (the “blocking adversary”) which in each step  $n$  computes a set  $B_n$ , the “blocking set”. For now, we assume this adversary knows witnesses for every step, i.e.,  $\vec{w}$  such that  $T^\ell(M_0, \vec{w}) = M_\ell$ . The “blocking sets” produced by  $\mathcal{B}$  will satisfy:

- Disjointness:** The blocking sets are disjoint:  $\forall i, j : i \neq j \implies B_i \cap B_j = \emptyset$ .
- Frequent Appearance:** In a random run from step  $n$  until step  $\ell$  the expected number of elements from  $B_n$  which occur in  $\mathcal{V}_\downarrow^{(i)}$  for  $i \geq n$  is at least  $(\ell - n)/2$ .



Formally:

$$\mathbb{E} \left[ \sum_{i=n}^{\ell} |B_n \cap \mathcal{V}_{\downarrow}^{(i)}| \right] > (\ell - n)/2.$$

We first argue that if we can prove the two properties then we are done. Assume *disjointness* and *frequent appearance*. By linearity of expectation and Gauss’ trick we get that

$$\mathbb{E} \left[ \sum_{n=1}^{\ell} \sum_{i=n}^{\ell} |B_n \cap \mathcal{V}_{\downarrow}^{(i)}| \right] > \sum_{n=1}^{\ell} (\ell - n)/2 = \frac{\ell(\ell - 1)}{4}.$$

By disjointness we get that

$$\sum_{n=1}^{\ell} \sum_{i=n}^{\ell} |B_n \cap \mathcal{V}_{\downarrow}^{(i)}| = \sum_{i=1}^{\ell} \sum_{n=1}^i |B_n \cap \mathcal{V}_{\downarrow}^{(i)}| = \sum_{i=1}^{\ell} |(\cup_{n=1}^i B_n) \cap \mathcal{V}_{\downarrow}^{(i)}| \leq \sum_{i=1}^{\ell} |\mathcal{V}_{\downarrow}^{(i)}|.$$

Combining the last two inequalities we get that

$$\mathbb{E} \left[ \sum_{i=1}^{\ell} |\mathcal{V}_{\downarrow}^{(i)}| \right] > \frac{\ell(\ell - 1)}{4}.$$

This shows that it happens with non-zero probability that

$$\sum_{i=1}^{\ell} |\mathcal{V}_{\downarrow}^{(i)}| > \frac{\ell(\ell - 1)}{4}.$$

Therefore there must exist  $i$  such that it happens with non-zero probability that  $|\mathcal{V}_{\downarrow}^{(i)}| > \frac{\ell(\ell - 1)}{4\ell}$ , which proves the lemma.

*The Blocking Adversary.* Let  $p$  be a polynomial which we specify below. The blocking adversary  $\mathcal{B}$  runs as follows.

1. Let  $M_0$  be the start state,  $\pi_0 = \epsilon$ , and  $\vec{w}$  a witness vector
2. For  $m = 1, \dots, \ell$  compute  $M_m = T(M_{m-1}, w_m)$
3. Let  $B_{<1} = \emptyset$  and for  $m = 1, \dots, \ell$  do:
  - (a) Query the random oracle on all points in  $B_{<m} = \cup_{i=1}^{m-1} B_i$
  - (b) Compute  $\pi_m = \mathbb{P}^{\mathcal{O}}(T, M_{m-1}, \pi_{m-1}, w_m)$
  - (c) Run  $\mathbb{V}^{\mathcal{O}}(T, M_0, M_m, \pi_m)$  and name its queries  $\mathcal{V}_{\downarrow}^{(m)}$
  - (d) For  $\iota = 1, \dots, p$  let  $\pi_m^{(\iota)} = \pi_m$  and for  $f = m, \dots, \ell$  do:
    - i. Run  $\mathbb{V}^{\mathcal{O}}(T, M_0, M_f, \pi_f^{(\iota)})$  and record its queries  $\mathcal{V}_{\downarrow}^{(f, \iota)}$
    - ii. Add  $\mathcal{V}_{\downarrow}^{(f, \iota)} \setminus B_{<m}$  to  $B_m$
    - iii. If  $f < \ell$  then compute  $\pi_{f+1}^{(\iota)} = \mathbb{P}^{\mathcal{O}}(T, M_f, \pi_f^{(\iota)}, w_{f+1})$

Call the experiment  $\text{HonExp}$ . Note that the blocking sets are disjoint by construction. We now prove frequent appearance by appealing to zero-knowledge.

*Likely/Unlikely Queries.* For  $m \geq 1$  let  $\mathcal{B}^m$  be the adversary running as  $\mathcal{B}$  except that it simulates 3(b) in iteration  $m$  instead of using the witness for this step and call it  $\text{SimExp}_m$ . Let  $S_m$  be the set of queries programmed by the simulator. We call a point  $q \in S_m$  *likely* if when  $\mathcal{B}_m$  runs forward from step  $m$  then  $q$  appears in  $\mathcal{V}_\downarrow^{(\geq m)} = \cup_{i=m}^\ell \mathcal{V}_\downarrow^{(i)}$  with probability at least  $(q|S_m|)^{-1}$  for a polynomial  $q$  specified below. Let  $L_m \subseteq S_m$  be the set of likely points. Conversely we call  $U_m = S_m \setminus L_m$  the *unlikely* points.

Since  $|U_m| \leq |S_m|$ , it follows by a union bound that if we do a random run, then the probability that an unlikely point is verified is low. More precisely,

$$\Pr \left[ U_m \cap \mathcal{V}_\downarrow^{(\geq m)} \neq \emptyset \right] \leq q^{-1}.$$

*Collecting All Likely Queries.* For all  $q$  we can set  $p$  to be a polynomial such that if  $\mathcal{B}^m$  does  $p$  random runs from step  $m$  on, then except with negligible probability the likely points are included in the blocking set  $B_m$ , as described now. Consider any likely point  $q$ . In a random run it appears with probability at least  $(q|S_m|)^{-1}$ . So if we do  $q|S_m|$  independent runs it appears in one of these except with probability about  $1/e$  as  $(1 - 1/n)^n \rightarrow 1/e$  with fast convergence. So if we run for instance  $\lambda q$  times, it appears except with probability about  $e^{-\lambda}$ . Then use that negligible probabilities are maintained by polynomial union bounds and that the size of  $S_m$  is polynomial. This gives us that the likely point will appear in some  $\mathcal{V}_\downarrow^{(f,\ell)}$ . It will therefore be added to the blocking set when the adversary adds  $\mathcal{V}_\downarrow^{(f,\ell)} \setminus B_{<m}$  to  $B_m$ . Namely, the query  $q$  will not be in  $B_{<m}$  as the adversary queried on all points in  $B_{<m}$  before the simulation step was run. So by *fresh reprogramming* no element from  $S_m$  is in  $B_{<m}$ , and all likely points are in  $S_m$  by definition.

Next we argue that we can pick  $q$  large enough such that:

$$\mathbb{E} \left[ \sum_{n \geq m} |U_m \cap \mathcal{V}_\downarrow^{(n)}| \right] \leq 1/2.$$

For  $q > 2\ell|S_m|$  it holds that a given unlikely point appears with probability at most  $1/q$ , by definition, and that when it does it contributes at most  $\ell$  to the sum (if it is in all verifier sets). So its contribution to the expected value is at most  $\ell/q = 1/2|S_m|^{-1}$ . Then use that  $U_n \subset S_n$  to see that there are at most  $|S_n|$  unlikely points and apply linearity of expectation.

*Putting the Pieces Together.* By Lemma 3 we have that

$$\mathbb{E} \left[ \sum_{n \geq m} |S_m \cap \mathcal{V}_\downarrow^{(n)}| \right] \geq \ell - m - \text{negl}(\lambda).$$

Combining the above two inequalities and  $L_m = S_n \setminus U_m$  we get that:

$$\mathbb{E} \left[ \sum_{n \geq m} |L_m \cap \mathcal{V}_\downarrow^{(n)}| \right] \geq \ell - m - \text{negl}(\lambda) - 1/2 \geq \ell - m - 1.$$

Using that  $L_m \subset B_m$  we obtain:

$$\mathbb{E} \left[ \sum_{n \geq m} \left| B_m \cap \mathcal{V}_{\downarrow}^{(n)} \right| \right] \geq \ell - m - 1.$$

This inequality holds in  $\text{SimExp}_m$ . Now run  $\mathcal{B}$  ( $\text{HonExp}$ ) instead of  $B_m$ . Then by a reduction to zero-knowledge we easily get that:

$$\mathbb{E} \left[ \sum_{n \geq m} \left| B_m \cap \mathcal{V}_{\downarrow}^{(n)} \right| \right] \geq \frac{\ell - m}{2}.$$

Namely, the value  $\sum_{n \geq m} |B_m \cap \mathcal{V}_{\downarrow}^{(n)}|$  can be computed in poly-time in both experiments. So, if  $\mathbb{E} \left[ \sum_{n \geq m} \left| B_m \cap \mathcal{V}_{\downarrow}^{(n)} \right| \right] \geq \ell - m - 1$  in the real world and  $\mathbb{E} \left[ \sum_{n \geq m} \left| B_m \cap \mathcal{V}_{\downarrow}^{(n)} \right| \right] < \frac{\ell - m}{2}$  in the simulation we can easily make a distinguisher which computes  $\sum_{n \geq m} |B_m \cap \mathcal{V}_{\downarrow}^{(n)}|$  and uses it to guess whether we simulated in step  $m$  or not. This completes the proof.  $\square$

*Proof. (Proof of Theorem 1).* By combining Lemma 1, Lemma 3, and Lemma 4 we conclude that any proof system for  $T_H$ , we can pick the number of steps  $\ell$  to be a large enough polynomial such that the proof system will have some verifier of some step  $i$  make at least  $\frac{\ell - 1}{4}$  queries to its random oracle. Therefore the verifier must have running time at least  $\frac{\ell - 1}{4} \notin \text{poly}(|T_H|, \lambda, \log \ell)$ . This proves the first part of the theorem.

The second part is proven by combining Lemma 2, Lemma 3, and Lemma 4 similarly for  $T_{pp}$ .  $\square$

*Remark 5 (Simulation in the presence of computational assumptions).* Despite its simplicity the impossibility result above is quite general, in particular it applies to any non-deterministic  $\mathcal{O}$ -IVC scheme where the simulator works by only programming the random oracle—even in the presence of arbitrary computational assumptions. In particular it applies to interactive zero-knowledge arguments compiled in the random oracle model, like Fiat-Shamir transformations.

## 5 On Proving Impossibility Without Zero-Knowledge

In the previous section we proved impossibility for proof systems which are zero-knowledge. We now explore what it would take to circumvent this result. Can we construct non-deterministic IVC in the random-oracle model which is *not* zero-knowledge. Towards this we prove impossibility of non-deterministic IVC in the random-oracle model with the following properties:

**Knowledge Soundness.** The proof is knowledge sound.

**Blackbox.** The knowledge extractor only has blackbox rewinding access to the prover.

**Structured Queries.** When the prover makes a query  $x$  to the random oracle, then with good probability it knows whether the query  $x$  was made already or whether it is the first time the random oracle is queried on  $x$ .

**Collision Intractability.** There exist collision intractable hash functions.

We know that it is possible to make blackbox knowledge sound proofs in the random oracle model, for instance Micali’s CS proofs. It is hard to imagine a world where it is reasonable to assume a random oracle, but where a family of collision intractable hash functions does not exist. Our result can therefore be interpreted as saying that it is the succinctness plus structured queries that give the impossibility.

We discuss the structure assumption briefly. First of all, this is clearly not a reasonable assumption about any proof system. It says that the prover has to “remember” previous queries using a small state. It is easy to make proof systems that do not have this property. For instance, in iteration  $r$  flip a uniformly random bit and query  $r$  if and only if the bit is 1. On the other hand, it seems hard to exploit such forgotten queries in a constructive way. The result therefore hints that if we want to circumvent Theorem 1 we need to come up with completely new ways to use a random oracle. To see this, note that when querying random oracles in a proof system one typically makes two types of queries. One can make a query on a fresh point to get a fresh “challenge” that the prover is not in control over *ala* the Fiat-Shamir transform. In this case it is crucial that the queried point is fresh such that the challenge is unknown until the time of query. Typically *provers* makes this type of query. One can also make a query to check the validity of a previous query, for instance when recomputing a hash path in a Merkle tree in the CS proofs. In this case one knows that the point on which the queries are made are not fresh, at least in an honest run of the proof system. Typically *verifiers* make this type of query. However, in an iterative proof system we can imagine that also provers make such queries, possibly to check previous provers or verifiers. We leave it as an open problem to determine whether there are proof systems without structured queries which allow to circumvent Theorem 1.

We proceed to the proof. A simple, yet central, component in our proof is a simple lemma which states that for polynomially long computations there will be polynomially long “stretches” of proof steps where no fresh query made during the proofs in the stretch is checked by the final verifier.

**Lemma 5 (Non-trivial  $\mathcal{O}$ -IVC Implies Unchecked Stretches).** *Let the running time of  $\mathbb{V}^{\mathcal{O}}(x, \pi)$  be bounded by a polynomial  $\psi \in \text{poly}(|\mathcal{R}|, \lambda, \log \ell)$ . Then for all lengths  $q \in \text{poly}(|\mathcal{R}|, \lambda, \log \ell)$  there exist large enough  $\ell \in \text{poly}(|\mathcal{R}|, \lambda)$  and a position  $p \in [1, \dots, \ell]$  such that  $\mathcal{P}_{\Delta}^{(p,q)} \cap \mathcal{V}_{\downarrow}^{(\ell)} = \emptyset$  with non-negligible probability. The position  $p$  may depend on  $\lambda$ .*

*Proof.* Let  $\ell$  be a free variable for now, we fix it later. Since  $|\mathcal{R}| \in \text{poly}(\lambda)$  it is sufficient to consider any constants  $a, b \in \mathbb{N}$  and thereby any polynomial

$q = \lambda^b(\log \ell)^c$ . We want to show that there exists  $d$  such that if we let  $\ell = \lambda^d$  then there exists a position  $p$  (which might be a function of  $\lambda$ ) such that

$$\Pr_{\lambda}[\mathcal{P}_{\Delta}^{(p,q)} \cap \mathcal{V}_{\downarrow}^{(\ell)} = \emptyset] = \text{negl}(\lambda),$$

where  $\Pr_{\lambda}$  denotes that the probability is taken over a random run with security parameter set to  $\lambda$ .

For any  $q$  as above we can consider  $e = b + 1$  and  $q' = \lambda^e$ . We have that  $q' > q$  for large enough  $\lambda$  as  $\ell = \text{poly}(\lambda)$ . So for large enough  $\lambda$  we have that  $\mathcal{P}_{\Delta}^{(p,q)} \subset \mathcal{P}_{\Delta}^{(p,q')}$ . It is therefore sufficient to consider any constant  $e \in \mathbb{N}$  and thereby any polynomial  $q' = \lambda^e$  and show that there exists  $d$  such that if we let  $\ell = \lambda^d$  then there exists a position  $p$  such that

$$\Pr_{\lambda}[\mathcal{P}_{\Delta}^{(p,q')} \cap \mathcal{V}_{\downarrow}^{(\ell)} = \emptyset] = \text{negl}(\lambda).$$

Now that  $q'$  does not depend on  $\ell$  we can for any  $\phi \in \text{poly}(\lambda)$  set  $\ell = q\phi$ . Then we have  $\phi$  disjoint stretches  $(1, q), (q + 1, q), \dots, (\ell - q + 1, q)$ . This by definition gives disjoint sets  $\mathcal{P}_{\Delta}^{(1,q)}, \mathcal{P}_{\Delta}^{(q+1,q)}, \dots, \mathcal{P}_{\Delta}^{(\ell-q+1,q)}$ .

Since the running time of  $\mathbb{V}^{\mathcal{O}}(x, \pi)$  is bounded by  $\text{poly}[|\mathcal{R}|, \lambda, \log \ell]$  it is also bounded by some  $\phi \in \text{poly}(\lambda)$  for large enough  $\lambda$  via the same arguments as above. So for large enough  $\lambda$  the verifier can make at most  $\psi$  queries to the oracle, i.e.,  $|\mathcal{V}_{\downarrow}^{(\ell)}| \leq \psi$ . So if we set  $\phi = 2\psi$ , then in any given run at most half the sets  $\mathcal{P}_{\Delta}^{(p,q)}$  enumerated above will contain an element from  $\mathcal{V}_{\downarrow}^{(\ell)}$ .

For each large enough  $\lambda$  this allows us to pick a fixed position  $p_{\lambda}$  such that for a random run  $\mathcal{P}_{\Delta}^{(p,q)}$  will contain an element from  $\mathcal{V}_{\downarrow}^{(\ell)}$  with probability at most  $1/2$ . For smaller  $\lambda$  simply let  $p_{\lambda} = 1$ . Now let  $p(\lambda) = \lambda$ . Then

$$\exists \lambda' \forall \lambda > \lambda' \Pr_{\lambda}[\mathcal{P}_{\Delta}^{(p,q')} \cap \mathcal{V}_{\downarrow}^{(\ell)} = \emptyset] \geq \frac{1}{2}$$

which is non-negligible. □

Note that the function  $p(\lambda)$  can be computed in non-uniform PPT in  $\lambda$  by a simple lookup table. This is enough for where we use the lemma as we consider non-uniform adversaries for simplicity. We could, however, also get impossibility for uniform adversaries. If we allow  $p$  to be randomized (and all subsequent proofs can handle a randomized  $p$ ), then we can simply set  $\ell$  as in the proof and do  $\lambda$  runs of the experiment. We can then let  $p(\lambda)$  be any position where  $\mathcal{P}_{\Delta}^{(p,q')} \cap \mathcal{V}_{\downarrow}^{(\ell)} = \emptyset$  happens with frequency at least  $\frac{1}{2}$ . It is easy to see that such a position exists and will have  $\Pr_{\lambda}[\mathcal{P}_{\Delta}^{(p,q')} \cap \mathcal{V}_{\downarrow}^{(\ell)} = \emptyset] \neq \text{negl}(\lambda)$  in a fresh run.

Before giving the full proof, we prove a warmup case (Lemma 6) to give the intuition of the proof up front. The lemma just says that if a long witness is hashed and then the witness extracted, then it is the original witnesses which is

extracted, or collision intractability is broken. We then later show how to exploit this to get impossibility by showing that it cannot be the case that the original witness is extracted.

We describe a class of adversaries  $\mathcal{A}_{H,\ell,\vec{w},\rho}^{(\cdot)}$  in Fig. 2. Let  $\mathcal{A}_\ell^{(\cdot)}$  denote the random variable describing  $\mathcal{A}_{H,\ell,\vec{w},\rho}^{(\cdot)}$ , where  $H$ ,  $\vec{w}$  and  $\rho$  are sampled at random. And let  $\vec{w}(\mathcal{A}_\ell^\mathcal{O})$  denote the witnesses used by this adversary when run with oracle  $\mathcal{O}$ .

**Lemma 6.** *There exists a PPT algorithm  $\mathbb{E}$  such that when  $\mathcal{O}$  is a random oracle and  $(H, M_\ell, \pi_\ell) \leftarrow \mathcal{A}_\ell^\mathcal{O}$  then  $\mathbb{E}^{\mathcal{A}_\ell^\mathcal{O}} = \vec{w}(\mathcal{A}_\ell^\mathcal{O})$  except with negligible probability.*

*Proof.* Since  $\mathcal{A}_\ell^\mathcal{O}$  internally runs an honest proof using  $\mathbb{P}$  we have that  $\mathbb{V}^\mathcal{O}(H, M_\ell, \pi_\ell) = \top$  except with negligible probability. So, by knowledge soundness we have that there exists a PPT extractor  $\mathbb{E}$  such that if we let  $\vec{w}' = \mathbb{E}^{\mathcal{A}_\ell^\mathcal{O}}$  then  $M_\ell = H^\ell(M_0, \vec{w}')$  except with negligible probability. Let  $\vec{w} = \vec{w}(\mathcal{A}_\ell^\mathcal{O})$ . We have by construction that  $M_\ell = H^\ell(M_0, \vec{w})$ . This implies that  $\vec{w}' = \vec{w}$  or  $(\vec{w}, \vec{w}')$  is a collision for  $H$ . It is therefore enough to prove that  $(\vec{w}, \vec{w}')$  is a collision for  $H$  with negligible probability. This follows from a simple reduction to collision intractability of  $H$  using the fact that  $\mathbb{E}$  is a fixed PPT algorithm and  $H$  is chosen at random after  $\mathbb{E}$  is fixed.  $\square$

The above simple case shows that if the proof system has knowledge soundness we can make the extractor extract the long witness  $\vec{w}$  from blackbox interaction with the adversary. The only way the extractor learns information is via the queries of the adversary to the random oracle. We now show that it is possible for  $\mathcal{A}$  to use a fake hardcoded oracle for a long stretch of the proof and still have the proof be accepted with good probability. This is because the verifier does not have queries enough to test a query from all proof steps. During this stretch the adversary will not query the real oracle  $\mathcal{O}$ . So there is no interaction with the extractor. Hence the extractor does not learn enough about the witness used during the stretch to be able to extract it. We now flesh out this intuition.

The adversary  $\mathcal{A}_{H,\ell,\vec{w},\rho}^\mathcal{O}$  has the following values hard-coded. A hash function  $H$ , the number of steps  $\ell$ , the witnesses  $\vec{w}$ , and a random tape  $\rho = (\rho_1, \dots, \rho_\ell)$  long enough to provide  $\mathbb{P}$  with randomness  $\ell$  times. Let  $\mathcal{O}$  denote the oracle used by the adversary. The adversary proceeds as follows.

1. Let  $M_0 = 0^\lambda$  and  $\pi_0 = \epsilon$ .
2. For  $i = 1, \dots, \ell$  compute  $M_i = H(M_{i-1}, w_i)$  and
 
$$\pi_i = \mathbb{P}^\mathcal{O}(H, M_{i-1}, \pi_{i-1}, w_i; \rho_i) .$$
3. Output  $(H, M_\ell, \pi_\ell)$ .

**Fig. 2.**  $\mathcal{A}_{H,\ell,\vec{w},\rho}^{(\cdot)}$

The transition function we look at is simply collision intractable hashing. We describe the class of transition functions  $\mathcal{T}$ . We assume we have a family of collision intractable hash functions  $\mathbf{H} : \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$ . The witnesses are given by  $\vec{w} \in (\{0, 1\}^\lambda)^\ell$ . The step function  $T$  is represented by a hash function  $\mathbf{H}$ . We always let  $M_0 = 0^\lambda$  and the step function is given by  $M_i = T(M_{i-1}, w_i) = \mathbf{H}(M_{i-1}, w_i)$ . Since  $M_0$  is fixed we drop it from the notation below.

The adversary  $\mathcal{A}_{\mathbf{H}, \ell, (p, q), \vec{w}^{\text{PRE}}, \vec{w}^{\text{POST}}, \rho, \tilde{\mathcal{O}}, \tilde{\mathcal{W}}, b}^\mathcal{O}$  has the following values hard-coded. A hash function  $\mathbf{H}$ , the number of steps  $\ell$ , a stretch  $(p, q)$ , the *pre-stretch witnesses*  $\vec{w}^{\text{PRE}} = (w_1, \dots, w_{p-1})$ , the *post-stretch witnesses*  $\vec{w}^{\text{POST}} = (w_{p+q}, \dots, w_\ell)$ , a random tape  $\rho$  long enough to provide  $\mathbb{P}$  with randomness  $\ell$  times, an oracle  $\tilde{\mathcal{O}} : \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$  called the *stretch oracle*, an oracle  $\tilde{\mathcal{W}} : \{0, 1\}^* \rightarrow (\{0, 1\}^\lambda)^q$  called the *witness oracle*, and a switch  $b \in \{0, 1\}$ . Let  $\mathcal{O}$  denote the oracle which the adversary has oracle access to. The adversary proceeds as follows.

1. Let  $M_0 = 0^\lambda$  and  $\pi_0 = \epsilon$ .
2. Let  $(w_1, \dots, w_{p-1}) = \vec{w}^{\text{PRE}}$ .
3. For  $i = 1, \dots, p-1$  compute  $M_i = \mathbf{H}(M_{i-1}, w_i)$  and

$$\pi_i = \mathbb{P}^\mathcal{O}(\mathbf{H}, M_{i-1}, \pi_{i-1}, w_i; \rho_i) .$$

4. Let  $\mathcal{P}_\downarrow^{\text{PRE}} = \mathcal{P}_\downarrow^{\cup(1, p-1)}$  be the queries from  $\mathbb{P}$  to  $\mathcal{O}$  in the above step. Let  $\mathcal{O}^{\text{STR}} = [\mathcal{P}_\downarrow^{\text{PRE}} \mapsto \mathcal{O}, \tilde{\mathcal{O}}]$ . For the  $i$ 'th query  $q_i \in \mathcal{P}_\downarrow^{\text{PRE}}$  in order of appearance in the execution let  $y_i = \mathcal{O}(q_i)$  be the reply given by  $\mathcal{O}$  to the query  $q_i$  in the above step, let  $h = |\mathcal{P}_\downarrow^{\text{PRE}}|$ , and define the *query tag*  $T = ((q_1, y_1), \dots, (q_h, y_h))$ .
5. Define the *stretch witnesses*  $\vec{w}^{\text{STR}} = (w_p, \dots, w_{p+q-1}) = \tilde{\mathcal{W}}(T, M_{p-1}, \pi_{p-1})$ .
6. For  $i = p, \dots, p+q-1$  compute  $M_i = \mathbf{H}(M_{i-1}, w_i)$  and

$$\pi_i = \mathbb{P}^{\mathcal{O}^{\text{STR}}}(\mathbf{H}, M_{i-1}, \pi_{i-1}, w_i; \rho_i) .$$

7. Let  $\text{used}_{p+q} = \text{used}(\mathbf{H}, M_{i-1}, \pi_{i-1}, w_i; \rho_i)$ .
8. Let  $\mathcal{P}_\downarrow^{\text{STR}} = \mathcal{P}_\downarrow^{\cup(p, p+q-1)} \setminus \mathcal{P}_\downarrow^{\cup(1, p-1)}$  be the queries from  $\mathbb{P}$  to  $\tilde{\mathcal{O}}$  in the above step.
9. Let  $\mathcal{O}_0^{\text{POST}} = [\mathcal{P}_\downarrow^{\text{PRE}} \mapsto \mathcal{O}, \mathcal{P}_\downarrow^{\text{STR}} \mapsto \tilde{\mathcal{O}}, \mathcal{O}]$ .
10. Let  $\mathcal{O}_1^{\text{POST}}$  be the following oracle.

$$\mathcal{O}_1^{\text{POST}}(q) = \begin{cases} \tilde{\mathcal{O}}(q) & \text{if } \text{used}_{p+q}(q) = \top \wedge q \notin \mathcal{P}_\downarrow^{\text{PRE}} \\ \mathcal{O}(q) & \text{otherwise} \end{cases}$$

11. For  $i = p+q, \dots, \ell$  compute  $M_i = \mathbf{H}(M_{i-1}, w_i)$  and

$$\pi_i = \mathbb{P}^{\mathcal{O}_b^{\text{POST}}}(\mathbf{H}, M_{i-1}, \pi_{i-1}, w_i; \rho_i) .$$

12. Output  $(\mathbf{H}, M_\ell, \pi_\ell)$ .

**Fig. 3.**  $\mathcal{A}_{\mathbf{H}, \ell, (p, q), \vec{w}^{\text{PRE}}, \vec{w}^{\text{POST}}, \rho, \tilde{\mathcal{O}}, \tilde{\mathcal{W}}, b}^\mathcal{O}$

We describe a class of adversaries in Fig. 3. In the proof we will need to go through some hybrids. For simplicity we provide a single adversary with some parameters (two oracles and a binary switch) allowing to produce all the hybrids. For the same purpose we define in Fig. 4 an experiment with two binary switches  $a$  and  $b$ .

Note that the oracle  $\mathcal{O}^{\text{STR}} = [\mathcal{P}_{\downarrow}^{\text{PRE}} \mapsto \mathcal{O}, \tilde{\mathcal{O}}]$  used during the stretch will send all fresh queries to the simulated oracle  $\tilde{\mathcal{O}}$  so it will make no new queries to the real oracle  $\mathcal{O}$ . Note that  $\mathcal{O}_1^{\text{POST}}$  is the oracle which behaves like the real random oracle  $\mathcal{O}$  except on queries which according to  $\text{used}_{p+q}$  were made while proving for the stretch witness. The queries  $\text{used}_{p+q}(\mathfrak{q}) = \top \wedge \mathfrak{q} \notin \mathcal{P}_{\downarrow}^{\text{PRE}}$  are those presumable made before the stretch ended and not before the stretch started. For such stretch queries it uses the simulated oracle  $\tilde{\mathcal{O}}$ .

The experiment  $\text{EXTEXP}_{\ell,(p,q),a,b}^{\mathcal{O}}$  runs as follows.

1. Pick  $H, \vec{w}^{\text{PRE}}, \vec{w}^{\text{POST}}, \rho$  at random.
2. Let  $\tilde{\mathcal{O}}_0$  and  $\tilde{W}_0$  be uniformly random functions from their domain. Let  $\tilde{\mathcal{O}}_1$  and  $\tilde{W}_1$  be pseudo-random functions over their domains, specified by uniformly random keys  $O, W \in \{0, 1\}^\lambda$ .
3. Let  $\mathcal{A}^{(\cdot)} = \mathcal{A}_{H,\ell,(p,q),\vec{w}^{\text{PRE}},\vec{w}^{\text{POST}},\rho,\tilde{\mathcal{O}}_a,\tilde{W}_a,b}^{(\cdot)}$ .
4. Let  $(H, M_\ell, \pi_\ell) \leftarrow \mathcal{A}^{\mathcal{O}}$ .
  - Let  $\mathcal{P}_{\downarrow}^{\text{STR}}$  denote the set  $\mathcal{P}_{\downarrow}^{\text{STR}}$  used inside  $\mathcal{A}$ . Similarly for other variables used by  $\mathcal{A}$  like  $\mathcal{O}_b^{\text{POST}}$ ,  $\text{used}_{p+q}$ , and  $\mathcal{P}_{\cup}^{(i,k)}$ .
  - Let NSF be the *no structure failure* event that it did *not* happen that  $\mathcal{O}_b^{\text{POST}}$  was queried on an  $\mathfrak{q}$  such that  $(\text{used}_{p+q}(\mathfrak{q}) = \top \text{ and } \mathfrak{q} \notin \mathcal{P}_{\cup}^{(1,p+1)})$  or  $(\text{used}_{p+q}(\mathfrak{q}) = \perp \text{ and } \mathfrak{q} \in \mathcal{P}_{\cup}^{(1,p+1)})$ .
5. Let  $\mathcal{O}_0 = \mathcal{O}$  and let  $\mathcal{O}_1 = \mathcal{O}_b^{\text{POST}}$  be the oracle used in  $\mathcal{A}^{\mathcal{O}}$ .
6. For  $c = 0, 1$  let  $J_c = \mathbb{V}^{\mathcal{O}_c}(H, M_\ell, \pi_\ell)$ .
  - Let  $\mathcal{V}_c = \mathcal{V}_{\downarrow}^{(\ell)}$  be the queries from  $\mathbb{V}^{(\cdot)}(H, M_\ell, \pi_\ell)$  to its oracle  $\mathcal{O}_c$  and let QFS $_c$  be the *query-free stretch* event that  $\mathcal{V}_c \cap \mathcal{P}_{\downarrow}^{\text{STR}} = \emptyset$ . Let QFS = QFS $_0$ .
  - Let VER $_c$  be the event that  $J_c = \top$ .
7. Let  $\vec{v} = \mathbb{E}^{\mathcal{A}^{(\cdot)}, \mathcal{O}}$  and let  $\vec{v}^{\text{PRE}} \parallel \vec{v}^{\text{STR}} \parallel \vec{v}^{\text{POST}} = \vec{v}$ , where  $|\vec{v}^{\text{PRE}}| = p - 1$  and  $|\vec{v}^{\text{STR}}| = q$ .
  - Let XTF be the *extraction of full witness* event that  $J_0 = \perp$  or  $\vec{v} = \vec{w}$ .
  - Let XTS be the *stretch extraction* event that  $\vec{v}^{\text{STR}} = \vec{w}^{\text{STR}}$ .

**Fig. 4.**  $\text{EXTEXP}_{\ell,(p,q),a,b}^{\mathcal{O}}$

We are particularly interested in the event  $\text{XTS}_{a=0,b=1}$ . This is the event that the random witness used during the stretch is correctly extracted when  $a = 0$  (such that a uniformly random oracle and uniformly random witness are used) and  $b = 1$  (such that  $\mathcal{O}_1^{\text{POST}}$  is used for the post-stretch part of the proof in step



11). Below we prove two lemmas about  $\text{XTS}_{a=0,b=1}$ , which allows us to give the following proof for Theorem 2.

*Proof (Proof of Theorem 2).* Under the premises of the theorem we can prove both Lemma 7 and Lemma 8, and these two lemmas are in contradiction.  $\square$

We first prove:

**Lemma 7.** *Under the premises of Theorem 2 the following holds. There exists a polynomial stretch length  $q \in \text{poly}(|\mathcal{R}|, \lambda, \log \ell)$  such that it is not possible to set the number of steps  $\ell$  to a polynomial and set the stretch position  $p$  such that*

$$\Pr[\text{XTS}_{a=0,b=1}] \geq 1/\lambda^{O(1)}.$$

*Proof.* Note that the stretch witness is picked uniformly at random using the oracle  $\tilde{W}$ . This means that it has entropy  $q|w|$ . The extractor needs to learn this many bits to extract the stretch witness. Let us be curious about from where it could learn this many bits of information.

Note that the stretch witness is computed as  $\tilde{w}^{\text{STR}} = (w_p, \dots, w_{p+q-1}) = \tilde{W}(T, M_{p-1}, \pi_{p-1})$ , where  $(T, M_{p-1}, \pi_{p-1})$  is the entire state of the proof up to step 5, including previous random-oracle queries and replies. This ensured that if the adversary is rewind behind step 5 then an independent, uniformly random stretch witness is picked. So we can ignore extractors rewinding behind step 5.

The extractor also learns no information *during* steps 5 and 6 as the prover queries the stretch oracle  $\tilde{O}$  during the stretch, not the real oracle. Hence the adversary does not interact with the extractor at all during steps 5 and 6.

We finally argue that interacting with the adversary after step 6 cannot leak the entire stretch witness. To see this note that the adversary only needs the post-stretch witness and  $\mathcal{O}_1^{\text{POST}}$  to run after step 6. The post-stretch witness is independent of the stretch witness, and we can represent  $\mathcal{O}_1^{\text{POST}}$  such that it does not contain all information about the stretch witness. Namely, we could indistinguishably for the extractor switch to  $a = 1$  such that  $\tilde{O}$  is pseudo-random and specified by a short key  $O$ . Note that we can then compute  $\mathcal{O}_1^{\text{POST}}$  given  $O$ ,  $\text{used}_{p+q}$ ,  $\mathcal{P}_\downarrow^{\text{PRE}}$ , and  $\mathcal{O}$ . The oracle  $\mathcal{O}$  and the queries  $\mathcal{P}_\downarrow^{\text{PRE}}$  were chosen before the stretch witness was chosen, so cannot depend on it. The key  $O$  is short so can only contain little information on the stretch witness. Finally, we assumed  $\text{used}_{p+q}$  can be computed from the state of the prover which we have assumed is  $\text{poly}(\lambda, \log \ell)$  for some fixed polynomial. So if we set  $q$  to some larger polynomial, then the state of the prover cannot contain the stretch witness. By making  $q$  large enough we can ensure that the stretch witness can be guessed only with negligible probability from the state of the adversary after step 6, which proves the lemma.  $\square$

We then prove the following contradicting lemma:

**Lemma 8.** *Under the premises of Theorem 2 the following holds. For all polynomials  $q \in \text{poly}(|\mathcal{R}|, \lambda, \log \ell)$  it is possible to set  $\ell$  to a polynomial and set  $p$  such that*

$$\Pr[\text{XTS}_{a=0,b=1}] \geq 1/\lambda^{O(1)}.$$

*Proof.* We will argue that when  $a = 0$  and  $b = 1$  then we can for all  $q$  set  $\ell$  such that  $J_0 = \top$  with polynomial probability. When this happens, then knowledge soundness gives us that the extraction  $\vec{v} = \mathbb{E}^{\mathcal{A}^{(\cdot)}, \mathcal{O}}$  must be correct and therefore  $\text{XTS}_{a=0, b=1}$  happened. Note that when  $b = 1$  then the adversary's proof is constructed with  $\mathcal{O}_1 = \mathcal{O}_1^{\text{POST}}$  but  $\mathbb{E}^{\mathcal{A}^{(\cdot)}, \mathcal{O}}$  extracts with  $\mathcal{O}$ . The key to the proof is to show that with polynomial probability it holds that 1) the proof constructed by the adversary verifies against  $\mathcal{O}_1^{\text{POST}}$  and that 2)  $\mathcal{O}_1^{\text{POST}}(q) = \mathcal{O}(q)$  for all queries made by the verifier. The first part would give  $J_1 = \top$  and the second part would give  $J_0 = J_1$ , and we would be done.

We first prove 2). Note that  $\mathcal{O}_0^{\text{POST}}$  and  $\mathcal{O}_1^{\text{POST}}$  only differ in which queries they send to the simulated oracle  $\tilde{\mathcal{O}}$ . The oracle  $\mathcal{O}_0^{\text{POST}}$  sends exactly the queries made for the first time in the stretch. The oracle  $\mathcal{O}_1^{\text{POST}}$  tries to do the same but relies on  $\text{used}_{p+q}$  correctly identifying queries made before step  $p+q$ . Under the assumption that NSF happens we have that  $\mathcal{O}_0^{\text{POST}} = \mathcal{O}_1^{\text{POST}}$  and by the assumption that the proof system is  $1/\lambda^{O(1)}$ -SOQ we can assume that NSF happens with polynomial probability. So let us proceed under the assumption that  $\mathcal{O}_0^{\text{POST}} = \mathcal{O}_1^{\text{POST}}$ . Then note that  $\mathcal{O}_0^{\text{POST}} = \mathcal{O}$  unless queried on a query made by the prover during the stretch and use Lemma 5 to set  $\ell$  large enough that the verifier with polynomial probability does not make such a query. This ensures that  $\mathcal{O}_1(q) = \mathcal{O}(q)$  for all queries made by the verifier, as desired.

To prove 1) we then need to argue that it is also the case that the proof made by the adversary verifies against  $\mathcal{O}_1^{\text{POST}}$ . But note that when  $\tilde{\mathcal{O}}$  is uniformly random, then  $\mathcal{O}_0^{\text{POST}}$  is just another uniformly random oracle. And therefore  $\mathcal{O}_1^{\text{POST}} = \mathcal{O}_0^{\text{POST}}$  is also a uniformly random oracle. So the proof of the adversary is a random, honestly generated proof relative to a uniformly random oracle  $\mathcal{O}_1^{\text{POST}}$ . Therefore, by completeness, it will also verify relative to  $\mathcal{O}_1^{\text{POST}}$ .  $\square$

## References

1. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018). <https://eprint.iacr.org/2018/046>
2. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53644-5\\_2](https://doi.org/10.1007/978-3-662-53644-5_2)
3. 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](https://doi.org/10.1007/978-3-662-44381-1_16)
4. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data, pp. 111–120 (2013). <https://doi.org/10.1145/2488608.2488623>
5. Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. Cryptology ePrint Archive, Report 2020/1618 (2020). <https://eprint.iacr.org/2020/1618>

6. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12551, pp. 1–18. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-64378-2\\_1](https://doi.org/10.1007/978-3-030-64378-2_1)
7. Chen, M., Chiesa, A., Spooner, N.: On succinct non-interactive arguments in relativized worlds. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022. LNCS, vol. 13276, pp. 336–366. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-07085-3\\_12](https://doi.org/10.1007/978-3-031-07085-3_12)
8. Chiesa, A., Liu, S.: On the impossibility of probabilistic proofs in relativized worlds, pp. 57:1–57:30 (2020). <https://doi.org/10.4230/LIPIcs.ITCS.2020.57>
9. Chiesa, A., Ojha, D., Spooner, N.: FRACTAL: post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 769–793. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45721-1\\_27](https://doi.org/10.1007/978-3-030-45721-1_27)
10. Chiesa, A., Tromer, E.: Proof-carrying data and hearsay arguments from signature cards, pp. 310–331 (2010)
11. Chiesa, A., Yogev, E.: Subquadratic SNARGs in the random oracle model. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 711–741. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84242-0\\_25](https://doi.org/10.1007/978-3-030-84242-0_25)
12. Chiesa, A., Yogev, E.: Tight security bounds for Micali's SNARGs. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13042, pp. 401–434. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-90459-3\\_14](https://doi.org/10.1007/978-3-030-90459-3_14)
13. Choudhuri, A.R., Jain, A., Jin, Z.: Non-interactive batch arguments for NP from standard assumptions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 394–423. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84259-8\\_14](https://doi.org/10.1007/978-3-030-84259-8_14)
14. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions, pp. 99–108 (2011). <https://doi.org/10.1145/1993636.1993651>
15. 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>
16. Haitner, I., Nukrai, D., Yogev, E.: Lower bound on SNARGs in the random oracle model. Cryptology ePrint Archive, Report 2022/178 (2022). <https://eprint.iacr.org/2022/178>
17. Kalai, Y.T., Paneth, O., Yang, L.: How to delegate computations publicly, pp. 1115–1124 (2019). <https://doi.org/10.1145/3313276.3316411>
18. Lipmaa, H., Pavlyk, K.: Gentry-wichs is tight: a falsifiable non-adaptively sound SNARG. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13092, pp. 34–64. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-92078-4\\_2](https://doi.org/10.1007/978-3-030-92078-4_2)
19. Micali, S.: CS proofs (extended abstracts), pp. 436–453 (1994). <https://doi.org/10.1109/SFCS.1994.365746>
20. Unruh, D.: Random oracles and auxiliary input. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 205–223. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-74143-5\\_12](https://doi.org/10.1007/978-3-540-74143-5_12)
21. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 1–18. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78524-8\\_1](https://doi.org/10.1007/978-3-540-78524-8_1)



# SNARGs and PPAD Hardness from the Decisional Diffie-Hellman Assumption

Yael Tauman Kalai<sup>1,3</sup>, Alex Lombardi<sup>2(✉)</sup>, and Vinod Vaikuntanathan<sup>3</sup>

<sup>1</sup> Microsoft Research, Cambridge, USA

<sup>2</sup> Simons Institute and UC Berkeley, Berkeley, USA  
alexlombardi@berkeley.edu

<sup>3</sup> MIT, Cambridge, USA

**Abstract.** We construct succinct non-interactive arguments (SNARGs) for bounded-depth computations assuming that the decisional Diffie-Hellman (DDH) problem is sub-exponentially hard. This is the first construction of such SNARGs from a Diffie-Hellman assumption. Our SNARG is also *unambiguous*: for every (true) statement  $x$ , it is computationally hard to find any accepting proof for  $x$  other than the proof produced by the prescribed prover strategy.

We obtain our result by showing how to instantiate the Fiat-Shamir heuristic, under DDH, for a variant of the Goldwasser-Kalai-Rothblum (GKR) interactive proof system. Our new technical contributions are (1) giving a  $TC^0$  circuit family for finding roots of cubic polynomials over a special family of characteristic-2 fields (Healy-Viola, STACS 2006) and (2) constructing a variant of the GKR protocol whose invocations of the sumcheck protocol (Lund-Fortnow-Karloff-Nisan, STOC 1990) only involve degree 3 polynomials over said fields.

Along the way, since we can instantiate the Fiat-Shamir heuristic for certain variants of the sumcheck protocol, we also show the existence of (sub-exponentially) hard problems in the complexity class PPAD, assuming the sub-exponential hardness of DDH. Previous PPAD hardness results required either bilinear maps or the learning with errors assumption.

## 1 Introduction

Succinct non-interactive arguments (SNARGs) [Mic94] are short, easy to verify, and computationally sound proofs that a statement  $x$  belongs to a potentially complex language  $L$ . In principle, one could hope to construct extremely efficient SNARGs for all NP languages; indeed, in the random oracle model, there exists a non-interactive<sup>1</sup> argument system for any  $L$  decidable in non-deterministic time  $T(n)$  with proof size  $\text{poly}(\lambda, \log T)$  and verification time  $\text{poly}(\lambda, \log T) + \tilde{O}(n)$ ,

<sup>1</sup> As is common, we consider arguments in the common reference string (CRS) model, where the reference string is set up in advance. Throughout this paper, our reference strings will be uniformly random without loss of generality.

where  $n = |x|$  and  $\lambda$  is a security parameter [Mic94]. Unfortunately, there are significant concerns about whether it is possible to construct such arguments based on falsifiable and preferably standard computational assumptions [Bar01, GK03, GW11, BBH+19].

In this work, we consider SNARGs for a *restricted* class of languages: those computable by *logspace-uniform* circuit families of bounded depth  $D$  (and arbitrary polynomial size  $S$ ). These were first studied in the interactive setting by Goldwasser, Kalai, and Rothblum [GKR08], who constructed (statistically sound) interactive proofs of size  $D \cdot \text{poly} \log(S)$  and verification time  $D \cdot \text{poly} \log(S) + \tilde{O}(n)$ . (We will henceforth refer to this as the GKR protocol.) Recently, a work of Jawale, Kalai, Khurana, and Zhang [JKKZ21] showed how to convert this proof system into a SNARG by instantiating the Fiat-Shamir heuristic [FS87, BR93] for the GKR protocol in the standard model (building upon [CCH+19]). Their SNARG is proved secure under the sub-exponential hardness of the learning with errors (LWE) assumption. Aside from [JKKZ21], SNARGs for (even unbounded depth) deterministic computation are now known from bilinear maps [KPY19, WW22, KLVW23], the polynomial hardness of LWE [CJJ22], and a combination of the decisional Diffie-Hellman (DDH) and Quadratic Residuosity (QR) assumptions [HJKS22, KLVW23].

**SNARGs from DDH.** In this work, we ask if SNARGs can be built from the DDH assumption *alone*. We answer this question in the affirmative for SNARGs for bounded-depth computations.

**Theorem 1.1.** *Assuming the sub-exponential hardness of DDH, there exist SNARGs for logspace-uniform depth- $D$ , size- $S$  computations. The SNARGs have proof size  $\text{poly}(\lambda, \log S) \cdot D$  and verification time  $\text{poly}(\lambda, \log S) \cdot (D + n)$ . The prover runs in time  $\text{poly}(\lambda, S)$ .*

As stated, our SNARGs achieve only non-adaptive soundness: that is, soundness holds for fixed inputs  $x$  to the circuit  $C$ .<sup>2</sup> By complexity leveraging (setting  $\lambda = n^{1/\epsilon}$  for an appropriate constant  $\epsilon$  independent of  $S$ ), we can achieve soundness that is adaptive with respect to the input  $x$ , at the cost of a communication complexity and verification time that are  $\text{poly}(n, \log S) \cdot D$ .

Moreover, our SNARGs are *unambiguous* [RRR16, CHK+19a], which means that they satisfy a form of soundness even for *true* statements: for  $x \in L$ , it is computationally hard to find an accepting proof for  $x$  *other than* the honestly generated proof guaranteed to exist by completeness. Unambiguous SNARGs were previously constructed in [KPY20, JKKZ21] but only known using either bilinear maps or LWE.

On a slightly more technical level, we show that (unambiguous) SNARGs for bounded depth can be built from a weaker generic primitive than was

---

<sup>2</sup> We note that the circuit  $C$  itself is always fixed in the protocol description.

known before: (lossy) correlation-intractable hash functions [CGH98, CCH+19, JKKZ21] supporting the complexity class  $\text{TC}^0$ . Previous work relied on a stronger form of correlation intractability (CI) supporting functions in  $\text{P}$  (or, implicitly,  $\text{NC}$ ). Since CI for  $\text{TC}^0$  was constructed based on DDH in [JJ21], this essentially implies Theorem 1.1. We discuss this in more detail in our technical overview (Sect. 1.1).

*Hardness in PPAD from DDH.* Closely tied to unambiguous SNARGs is the problem of showing the *cryptographic hardness* of the complexity class PPAD [Pap94, CHK+19a]. PPAD is a complexity class arising from computational game theory that famously includes finding a Nash equilibrium of bimatrix games as a complete problem [DGP09, CDT09]. The work of Choudhuri et al. [CHK+19a] showed that instantiating the Fiat-Shamir heuristic for (many variants of) the *sumcheck* protocol [LFKN90] suffices to establish PPAD-hardness.

In this work, towards instantiating Fiat-Shamir for the GKR protocol, we show how to instantiate Fiat-Shamir for variants of the sumcheck protocol that can be plugged into the framework of [CHK+19a]. Thus, we obtain PPAD-hardness from the sub-exponential DDH assumption.

**Theorem 1.2.** *Assuming that DDH is sub-exponentially hard, the complexity class PPAD contains problems that are sub-exponentially hard on average.*

Again, we prove Theorem 1.2 by showing that lossy correlation intractable hash functions for  $\text{TC}^0$  suffice to construct the non-interactive sumcheck protocol.

In the rest of this introduction, we give a brief overview of our techniques for proving Theorem 1.1 and Theorem 1.2.

## 1.1 Technical Overview

We begin by discussing our contributions on applying the Fiat-Shamir heuristic to the sumcheck protocol [LFKN90]. We first recall the sumcheck protocol.

*The Sumcheck Protocol.* In the sumcheck protocol, the prover and verifier begin with a degree<sup>3</sup>- $d$  polynomial  $f(x_1, \dots, x_n)$  in  $n$  variables over some finite field  $\mathbb{F}$ . The prover wants to convince the verifier of the value of the *sum*  $y = \sum_{a \in \{0,1\}^n} f(a)$ , where the sum is taken over  $\mathbb{F}$ . This is accomplished by the following interactive protocol:

- The prover sends the univariate polynomial

$$g(x) = \sum_{a_2, \dots, a_n \in \{0,1\}} f(x, a_2, \dots, a_n).$$

---

<sup>3</sup> Here and below, by “degree” we refer to individual degree: a multivariate polynomial has individual degree  $\leq d$  if it has degree  $\leq d$  in each variable.

- The verifier checks that  $g(0) + g(1) = y$ . If so, it samples a uniformly random  $\beta \leftarrow \mathbb{F}$  and sends  $\beta$  to the prover.
- The prover and verifier recursively execute the sumcheck protocol with respect to the polynomial  $f_\beta(x_2, \dots, x_n) = f(\beta, x_2, \dots, x_n)$  and value  $y_\beta = g(\beta)$ .
- In the base case, the verifier simply evaluates  $f(\beta_1, \dots, \beta_n)$ , which it can do given a circuit for  $f$ .

As shown in [CCH+19, JKKZ21], this protocol satisfies what is called unambiguous *round-by-round soundness*. Concretely, what this means (for this protocol) is that once the polynomial  $g$  is fixed by the prover, if  $g$  is *not* the correct “partial sum” polynomial, then with high probability over the choice of  $\beta$ , the prover and verifier will recurse on a false statement. Note also that if the statement  $(f, y)$  is false but  $g$  is the correct polynomial w.r.t.  $f$ , then the verifier will immediately reject.

*Removing Interaction via Fiat-Shamir.* In this work, we want a *non-interactive* variant of the sumcheck protocol, which we obtain by instantiating the Fiat-Shamir heuristic [FS87, BR93] for the sumcheck protocol. Concretely, this means that we will have  $n$  hash functions  $h_1, \dots, h_n$  sampled from a hash family  $\mathcal{H}$ , and the  $i$ -th verifier challenge  $\beta_i$  is instead computed as a hash  $h_i(f, y, g_1, \beta_1, \dots, g_i)$  of the transcript so far.

Following the *bad challenge function* paradigm of [CCH+19], we call a challenge  $\beta_i$  *bad* for  $(f, y, g_1, \beta_1, \dots, g_i)$  if (1)  $g_i$  is *not* the correct polynomial  $g_i^* = \sum_{a_{i+1}, \dots, a_n} f(\beta_1, \dots, \beta_{i-1}, x, a_2, \dots, a_n)$  and (2) the resulting recursive claim  $(f_{\beta_1, \dots, \beta_i}, g_i(\beta_i))$  is *true*. In the case of the sumcheck protocol, the set of all bad  $\beta$  is precisely the set of *roots* of the polynomial  $g_i(x) - g_i^*(x)$ . Note that there are at most  $d$  such roots, as  $g_i(x) - g_i^*(x)$  is a nonzero polynomial of degree at most  $d$ . Thus, letting  $F_1^{(i)}, \dots, F_d^{(i)}$  denote functions where  $F_j^{(i)}$  maps  $(f, \beta_1, \dots, \beta_{i-1}, g_i)$  to the  $j$ th root of  $g_i - g_i^*$  in  $\mathbb{F}$  (if one exists), we know by [CCH+19] that the resulting non-interactive protocol is sound if each  $h_i$  is correlation-intractable (CI) [CGH98, CCH+19] for  $F_1^{(i)}, \dots, F_d^{(i)}$ .

Recall that a hash function family  $\mathcal{H}$  is CI for a relation  $R$  (generalizing the case of a function  $f$ ) if given  $h \leftarrow \mathcal{H}$  it is computationally hard to find an input  $\alpha$  such that  $(\alpha, h(\alpha)) \in R$ . There has been much recent progress on constructing CI hash functions based on standard cryptographic assumptions (e.g. [CCH+19, PS19, BKM20, JJ21, HLR21]). The construction relevant to us in this work is that of [JJ21], which built a CI hash function family supporting functions computable in the complexity class  $\text{TC}^0$  (constant-depth threshold circuits) based on sub-exponential DDH.

Thus, in order to use the [JJ21] hash function family, we ask: **what is the computational complexity of the bad challenge functions  $F_j^{(i)}$ ?**

Naïvely, it is not even clear that the  $F_j^{(i)}$  functions are in  $\text{P}$ , because even computing the polynomial  $g_i^*$  (as a function of  $f, \beta_1, \dots, \beta_{i-1}$ ) seems to require time  $2^{n-i}$ . However, following [JKKZ21], if the functions  $h_1, \dots, h_{i-1}$  are *lossy* [PW08], we can *guess* the challenges  $\beta_1, \dots, \beta_{i-1}$  in advance and non-uniformly hard-wire the polynomial  $g_i^*$  in our security reduction (incurring a

sub-exponential security loss from guessing  $\beta_1, \dots, \beta_{i-1}$ ). That is, we will actually define each  $h_i$  to be the *composition* of a [JJ21] hash function with a lossy trapdoor function family (LTDF). The resulting composition will still be CI for  $\text{TC}^0$  circuits provided that *inversion* of a LTDF can be computed in  $\text{TC}^0$ , which we observe (see Sect. 2.2) is possible for a simple modification of standard constructions [PW08, FGK+10].

### 1.1.1 The Circuit Complexity of Root-Finding

Finally, we arrive at the first of two main challenges in this work. With the setup so far, we have reduced the problem to achieving correlation intractability for a circuit class that has the power to find roots of univariate polynomials over a field  $\mathbb{F}$  (and some additional  $\text{TC}^0$  operations). If root-finding over finite fields were known to be in  $\text{TC}^0$ , we would be done! Unfortunately, standard root-finding algorithms [Ber70, Rab80, CZ81] are *not* known to be implementable in  $\text{TC}^0$ . Indeed, it is clear that some care is required: if  $p$  is a large (size  $2^\lambda$ ) prime, finding roots of even *degree* 1 polynomials over  $\mathbb{F}_p$  is at least as hard as computing  $\text{mod } p$  inverses  $a \mapsto a^{-1} \pmod{p}$ , which is not known to be in  $\text{TC}^0$ .

Thus, it is clear that to have any hope of root-finding in  $\text{TC}^0$ , one must carefully choose the field  $\mathbb{F}$ . In this work, we make use of a special characteristic-2 field ensemble  $\mathbb{K} = \{\mathbb{K}_n\}$  studied by Healy and Viola, henceforth referred to as the Healy-Viola field ensemble, over which many field operations (including the inversion map  $a \mapsto a^{-1}$ ) are known to be in  $\text{TC}^0$  [HV06]. In this work, we show:

**Lemma 1.1 (informal, see Theorem 3.4).** *There is a  $\text{TC}^0$  circuit family that finds all roots of cubic (degree  $d = 3$ ) univariate polynomials over the Healy-Viola field ensemble.*

We emphasize that the algorithm in Lemma 1.1 only finds roots that lie in the ground field  $\mathbb{K}$ , *not* (necessarily) roots that lie in extensions<sup>4</sup> of  $\mathbb{K}$ . Combining Lemma 1.1 with our discussion so far, we obtain the following result:

**Theorem 1.3 (informal).** *Fiat-Shamir for degree-3 sumcheck protocols over the Healy-Viola field ensemble is sound using hash functions that are Lossy CI for  $\text{TC}^0$ , and is therefore instantiable under sub-exponential DDH.*

That is, by carefully instantiating the field ensemble and designing a special-purpose root-finding algorithm, we show how to use the Jain-Jin hash function family [JJ21] to achieve a non-interactive sumcheck for *degree three* polynomials. This is a very limited form of non-interactive sumcheck, but we next show how to leverage this limited form of sumcheck to prove Theorems 1.1 and 1.2. Finally, at the end of this overview we sketch a proof of Lemma 1.1, which is one of our main technical contributions.

---

<sup>4</sup> In fact, our algorithm finds all roots that lie in the unique degree-2 extension of  $\mathbb{K}$  but not its algebraic closure.



### 1.1.2 PPAD Hardness with Degree-2 Sumchecks

First, we show that PPAD-hardness can be established making use of our non-interactive sumcheck protocol for polynomials of degree as low as 2!

To employ the framework of [CHK+19a], all that we require is that our sumcheck protocol can be used to prove membership in a NP-hard language. In [CHK+19a], this is accomplished by using sumcheck over a *large* characteristic field as an argument system for #SAT.

Since our non-interactive sumcheck works over a characteristic-2 field, we instead start with  $\oplus$ SAT, the computational problem of counting the *parity* of the number of satisfying assignments of a SAT formula. This problem is also NP-hard under randomized reductions [VV85]. Given such a SAT formula  $\phi$ , this parity can then be expressed as a sumcheck problem over  $\mathbb{K}$ :

$$\oplus\text{SAT}(\phi) = \sum_{a_1, \dots, a_n \in \{0,1\}} \phi(a_1, \dots, a_n).$$

Moreover,  $\phi$  can be arithmetized so that it is represented by a polynomial-size arithmetic circuit over  $\mathbb{F}_2 \subset \mathbb{K}$ . In order for our non-interactive sumcheck protocol to be applicable, we would need the individual degree of this arithmetization to be at most 3. Given that we are doing a “standard” arithmetization, what is the individual degree of  $\phi$ ? If  $\phi$  is a CNF, this is nothing more than the maximum number of clauses that an individual variable appears in.

Conveniently, it is known that  $\oplus$ SAT on arbitrary formulas reduces to  $\oplus$ SAT on CNFs (which are not 3-regular) where each variable occurs in at most three clauses [Tov84]. This suffices to establish Theorem 1.2 by invoking Theorem 1.3 and [CHK+19a] with respect to a degree-3 sumcheck protocol.

*Note on Adaptivity.* In order to obtain Theorem 1.2, following [JKKZ21], we need a non-interactive sumcheck protocol satisfying a form of soundness referred to as “prefix-adaptive soundness” (an intermediate notion between non-adaptive and adaptive soundness). Our non-interactive sumcheck protocol satisfies this form of soundness exactly as in [JKKZ21].

*Variable-Extended Formulations.* While we have already proved Theorem 1.2, we will give a slightly different second proof, since this will be a crucial step in proving Theorem 1.1. Specifically, we will prove Theorem 1.2 by invoking a degree 2 (rather than 3) sumcheck protocol. To do this, we start with the sumcheck problem above with respect to (the standard poly-size arithmetization of)  $\phi$ . The individual degree of  $\phi$  may be very large; nevertheless, we will show that  $\oplus\text{SAT}(\phi)$  can be expressed as a *different* degree 2 sumcheck.

We accomplish this using a Cook–Levin style reduction, in which we introduce new variables  $y_1, \dots, y_m$  that are supposed to represent a *wire assignment* of the NAND-circuit computing  $\phi$ . Concretely, consider the polynomial  $f$  in

$n + m$  variables defined as

$$f(x_1, \dots, x_n, y_1, \dots, y_m) = y_m \cdot \prod_{(i,j,k) \in \text{Gates}(\phi)} (y_i + y_j y_k) \prod_{i=1}^n \left( x_i \prod_{j \in S_i} y_j + (1 - x_i) \prod_{j \in S_i} (1 - y_j) \right),$$

where for every  $i \in [n]$ ,  $S_i \subset [m]$  is defined to be the subset of leaf vertices in  $\phi$  that are assigned the input  $x_i$ . In words,  $f$  (arithmetically) checks that (i) the output wire  $y_m$  is 1, (ii) each NAND gate is computed correctly, and (iii) the leaf variables were all assigned with respect to a consistent  $x \in \{0, 1\}^n$ . For boolean inputs,  $f(x_1, \dots, x_n, y_1, \dots, y_m)$  is thus either zero or equal to  $\phi(x_1, \dots, x_n)$  (which happens for one “consistent” assignment to  $y$ ). Therefore,

$$\sum_{a_1, \dots, a_n \in \{0,1\}} \phi(a_1, \dots, a_n) = \sum_{\substack{a_1, \dots, a_n \in \{0,1\} \\ b_1, \dots, b_m \in \{0,1\}}} f(a_1, \dots, a_n, b_1, \dots, b_m),$$

so computing  $\oplus\text{SAT}(\phi)$  reduces to an  $f$ -sumcheck. Finally, note that  $f$  has individual degree 2! Indeed, it is linear in the  $x_i$ , quadratic in the non-leaf  $y_j$  (as they are each used in two gates of  $\phi$ ), and quadratic in the leaf  $y_j$  (each is used in one gate of  $\phi$  and has a linear dependence in the “input encoding” part of  $f$ ).

In general, we say that the above transformation produces a “variable-extended formulation” of a boolean formula  $\phi$  (see Definition 5.1), and this is a key step in proving Theorem 1.1.

### 1.1.3 SNARGs via Degree 3 Sumchecks

Armed with our new approach of constructing variable-extended formulations of sumcheck polynomials, we proceed to sketch our proof of Theorem 1.1. We prove Theorem 1.1 by choosing a suitable variant of the [GKR08] protocol, modifying it to rely only on degree 3 sumchecks, and then (essentially<sup>5</sup>) applying Theorem 1.3.

At a high level, the [GKR08] protocol proves that  $C(x) = y$  for a logspace-uniform depth  $D$ , size  $S$  circuit  $C$  by iteratively producing pairs of claims about a multilinear extension encoding of each layer  $L_i$  of the computation tableau of the circuit (when evaluated on input  $x$ ). That is, each  $L_i = L_i(x) \in \{0, 1\}^S$  is a string containing the value of all level  $i$  vertices in the evaluation of  $C(x)$ , and  $L_i$  is interpreted as a function  $\ell_i : \{0, 1\}^{\log S} \rightarrow \{0, 1\}$ , which can then be extended to a multilinear function  $\hat{\ell}_i : \mathbb{K}^{\log S} \rightarrow \mathbb{K}$ . The GKR protocol begins with an evaluation claim about  $\hat{\ell}_D$  (the end of the computation) and ends with a pair of evaluation claims about  $\hat{\ell}_0$ ; since the input layer of  $C$  has only width  $n$  (rather than, say,  $S/D$ )  $\hat{\ell}_0$  can be evaluated in  $O(n)$  field operations and thus can be checked by the verifier.

<sup>5</sup> The variant of [GKR08] that we use actually runs pairs of sumcheck protocols in parallel with shared verifier randomness (as is done in [Mei13, JKKZ21]), but this detail does not substantially change the proof.

The key step is to understand how to *reduce* claims about  $\hat{\ell}_i$  to claims about  $\hat{\ell}_{i-1}$ ; this boils down to the “sumcheck-friendly” equation which writes  $\ell_i(a)$  as

$$\sum_{b,c \in \{0,1\}^{\log S}} \left[ \chi_{\text{add}}^{(i)}(a,b,c) \left( \ell_{i-1}(b) + \ell_{i-1}(c) \right) + \chi_{\text{mult}}^{(i)}(a,b,c) \left( \ell_{i-1}(b)\ell_{i-1}(c) \right) \right],$$

where  $\chi_{\text{add}}, \chi_{\text{mult}}$  are the *gate indicator functions* that take as input three wire labels  $(a, b, c)$  for the circuit and indicates whether an addition (respectively, multiplication) gate is present at these three wires. This equation can then be extended multilinearly to a similar equation relating  $\hat{\ell}_i$  to  $\hat{\ell}_{i-1}$ .

Given this summary of the GKR protocol, the key question for us is as follows: what is the degree of the sumcheck polynomials? By inspection, it turns out that this degree is one more than the degree of the arithmetizations of  $\chi_{\text{add}}, \chi_{\text{mult}}$ . Naively, their degree may be up to  $O(\log S)$  (i.e., the number of leaves in the boolean formulas for  $\chi_{\text{add}}, \chi_{\text{mult}}$ ), but by using *variable-extended formulations* of these polynomials, we can reduce this degree to 2 (at the cost of adding  $O(\log S)$  auxiliary variables to the sumcheck). Note that it is crucial for *prover efficiency* that we only add  $O(\log S)$  (rather than  $\text{poly} \log S$ ) new variables, as the prover’s running time is exponential in this number of variables. We show that an appropriate extended formulation exists making use of an analysis due to Goldreich [Gol18] of  $\chi_{\text{add}}, \chi_{\text{mult}}$ .

In total, this results in a GKR protocol variant relying on degree 3 sumchecks over  $\mathbb{K}$ , and thus we can instantiate Fiat-Shamir for this protocol based on sub-exponential DDH.

### 1.1.4 Cubic Root Finding: Proving Lemma 1.1

Finally, we sketch a proof of Lemma 1.1, which states that roots of cubic polynomials over Healy-Viola fields  $\mathbb{K}$  can be computed in  $\text{TC}^0$ . We will *not* resort to general-purpose root-finding algorithms [Ber70, Rab80, CZ81] (which all have a high-depth iterative nature) but instead turn to *explicit formulae* for roots of low degree polynomials. We show that these explicit formulae can be converted into low-depth algorithms.

First, let us consider the degenerate cases of linear and quadratic polynomials.

- Root-finding for linear polynomials is equivalent to solving a linear equation over  $\mathbb{K}$ , which reduces to multiplication and inversion over  $\mathbb{K}$ . These operations were shown to be in  $\text{TC}^0$  in [HV06].
- Since  $\mathbb{K}$  has characteristic 2, root-finding for quadratic polynomials reduces to finding roots of polynomials of the form  $x^2 + c$  and  $x^2 + x + c$ .<sup>6</sup> Then:
  - The polynomial  $x^2 + c$  always has a double root of  $c^{|\mathbb{K}|/2}$ ,<sup>7</sup> which can be computed in  $\text{TC}^0$  via low-depth exponentiation [HV06].

<sup>6</sup> This follows from the fact that  $az^2 + bz + c = 0 \iff (a/b \cdot z)^2 + (a/b \cdot z) + a/b^2 \cdot c = 0$ .

<sup>7</sup> This is the case since in characteristic 2 fields,  $-\alpha = \alpha$  for all  $\alpha$ .

- The polynomial  $x^2 + x + c$  has roots given by an explicit formula as a function of  $c$  related to its orbit  $\{c, c^2, c^4, \dots, c^{2^{n-1}}\}$  under the Frobenius map  $\alpha \mapsto \alpha^2$ . The form depends on the order of two dividing  $\log |\mathbb{K}|$  (which turns out to be 1 for the Healy–Viola fields) and is given implicitly in our proof of Theorem 3.3.

*Passing to a Quadratic Extension of  $\mathbb{K}$ .* We note that [CJJ21] also proves that quadratic root-finding in  $\mathbb{K}$  is in  $\text{TC}^0$  with a different approach; however, we give a more powerful algorithm that actually finds roots of this polynomial in an explicit quadratic extension  $\mathbb{L} \supset \mathbb{K}$  (even when no roots in  $\mathbb{K}$  exist). This more powerful algorithm is necessary to prove the cubic case of Lemma 1.1.

In order for this to make sense, we must be able to construct  $\mathbb{L}$  in a way so that operations in  $\mathbb{L}$  are similarly efficient to operations in  $\mathbb{K}$ . Fortunately, we are able to do this with a careful construction, noting that one way to construct a quadratic extension of  $\mathbb{K}$  is to add to it a solution to the equation  $x^2 + x + \omega = 0$ , where  $\omega$  is an explicit cube root of unity in  $\mathbb{K}$ . Since  $\omega$  alone generates a constant-size field, this greatly simplifies the problem of giving efficient algorithms for operations over  $\mathbb{L}$ . We show in Theorem 3.2 that all of the relevant field operations on  $\mathbb{L}$  are in  $\text{TC}^0$ , which requires opening up the [HV06] construction and extending their analysis to  $\mathbb{L}$ .

*The Cubic Case.* Finally, we compute roots of *cubic* polynomials over  $\mathbb{K}$  using an algorithmic variant of the cubic formula [Lag70] over characteristic 2 fields. At a high level, the characteristic 2 cubic formula reduces computing roots of a cubic polynomial (given its coefficients), modulo basic operations, to the following tasks:

1. Finding roots of a related *quadratic* polynomial defined over  $\mathbb{K}$ .
2. Computing the *cube root* map  $\alpha \mapsto \alpha^{1/3}$  (modulo cube roots of unity).
3. Solving a constant-size linear system involving these cube roots.

In Sect. 3, we show that all of these procedures are computable in  $\text{TC}^0$  and thus roots of all cubic polynomials can be computed in  $\text{TC}^0$ .

One important subtlety is that the roots computed in (1) above are not necessarily in  $\mathbb{K}$ , but in the quadratic extension  $\mathbb{L}$ ; relatedly, (2) requires computing cube roots of elements of  $\mathbb{L}$ . One must be careful to argue that (in our setting) the cubic formula algorithm does *not* have to pass into a degree 6 (or degree 3) extension of  $\mathbb{K}$ , which we have not explicitly constructed.

For a full explanation/proof of Lemma 1.1, we refer the reader to Sect. 3 (Theorem 3.4).

## 1.2 Related Work

*Fiat-Shamir in the Standard Model.* Over the last few years, a line of work including [CCR16, KRR17, CCRR18, HL18, CCH+19, PS19, BKM20, JJ21, HLR21, CJJ21, CJJ22, HJKS22] and many others has studied the instantiability

of the Fiat-Shamir heuristic using concrete, efficiently computable hash function families. Starting with the work of [CCH+19], there have been instantiations based on standard cryptographic assumptions (initially the learning with errors assumption [CCH+19,PS19]). The work of [JJ21] constructed NIZKs for NP under the sub-exponential DDH assumption by building a hash family that is correlation-intractable for  $TC^0$  functions from sub-exponential DDH. Beginning with the works of [CCH+19,JKKZ21], applying Fiat-Shamir to the [GKR08] protocol (to construct SNARGs in the standard model) has been explicitly studied, including a construction based on sub-exponential LWE [JKKZ21]. Finally, more recently the Fiat-Shamir approach has been used to build succinct *batch arguments* for NP [CJJ21,CJJ22,HJKS22] which in turn imply SNARGs for P [CJJ22,KVZ21].

*SNARGs Without Fiat-Shamir.* There have additionally been constructions of SNARGs for P that do not rely on the Fiat-Shamir heuristic [KPY19,GZ21,WW22,KLVW23], all of which currently rely on some form of cryptographic bilinear maps.

*Cryptographic Hardness of PPAD.* Establishing hardness in PPAD based on cryptographic assumptions has also received considerable attention, including [BPR15,GPS16,CHK+19a,CHK+19b,EFKP20,LV20,KPY20,BCH+22]. Previously, PPAD-hardness was known under the following sets of assumptions:

- Polynomially secure functional encryption [BPR15,GPS16], which can be built by a particular combination of three concrete assumptions [JLS21],
- Super-polynomial hardness of a falsifiable assumption on bilinear maps [KPY20],
- The sub-exponential LWE assumption [JKKZ21], and
- A combination of (polynomially-secure) LWE and the (polynomial) hardness of iterated squaring modulo a composite [BCH+22].

## 2 Preliminaries

### 2.1 Cryptographic Groups

Let  $\mathbb{G} = \{\mathbb{G}_\lambda\}$  be a group ensemble, indexed by a security parameter  $\lambda$ , such that group operations (and testing equality) can be computed in time  $\text{poly}(\lambda)$ .

**Definition 2.1 (Decisional Diffie-Hellman Assumption).** *We say that the decisional Diffie-Hellman (DDH) assumption with time  $T$  and advantage  $\mu$  holds for  $\mathbb{G}$  if any  $T(\lambda)$ -time algorithm  $\mathcal{A}(\cdot)$  has advantage at most  $\mu$  in distinguishing a random “DDH-tuple”  $(g, g^x, g^y, g^{xy})$  from a tuple  $(g, g^x, g^y, g^z)$  (for uniformly random  $x, y, z$ ).*

In this paper, we work exclusively with cryptographic groups satisfying the following conditions:<sup>8</sup>

<sup>8</sup> These groups will be used to instantiate the lossy trapdoor function component of a lossy CI hash family; the CI component does not have to satisfy all of these properties (but DDH must still be sub-exponentially hard).

1. *Iterated group multiplication*  $g_1, g_2, \dots, g_t \mapsto \prod_{i=1}^t g_i$  can be computed by a  $\text{poly}(\lambda, t)$ -size, *low-depth* circuit family. As in [JJ21], there are two flavors of results: one requires  $\text{TC}^0$  circuits (which exist for, e.g., subgroups of  $\mathbb{Z}_q^\times$ ), while the other requires threshold circuits (with unbounded fanin) of depth  $o(\log \lambda)$  (which exist for standard elliptic curve groups [JJ21]). In the latter case, we will use complexity leveraging: re-define the group security parameter to be  $\kappa = \text{poly} \log \lambda$ , so that DDH remains polynomially hard and iterated multiplication can be computed in depth  $o(\log \log \lambda)$ .
2. The DDH assumption holds with inverse-subexponential  $\mu = 2^{-\lambda^\epsilon}$  for some constant  $\epsilon > 0$ . If iterated multiplication requires superconstant-depth threshold circuits, then we require the assumption to hold for  $T = 2^{\lambda^\epsilon}$  (so that we can complexity leverage as above), while if iterated multiplication has  $\text{TC}^0$  circuits, then we only require the assumption to hold for  $T = \text{poly}(\lambda)$ .
3. Letting  $M$  denote a uniformly random  $n(\lambda) \times n(\lambda)$  matrix (for  $n(\lambda) = \text{poly}(\lambda)$ ) modulo  $N = |\mathbb{G}|$ , we have that  $M$  is invertible with probability  $1 - \text{negl}(\lambda)$ . This holds immediately for prime-order groups or groups with order  $N$  that have no polynomial-size prime divisors.

As discussed in [JJ21], properties (1) and (2) are satisfied (that is, DDH is plausible) by common examples such as (subgroups of)  $\mathbb{Z}_q^\times$  or groups of  $\mathbb{F}_q$ -points of elliptic curves.

## 2.2 Lossy Trapdoor Functions

Lossy trapdoor functions were first defined and constructed in an influential work of Peikert and Waters [PW08]. Loosely speaking, a lossy trapdoor function family contains two types of functions: injective ones and lossy ones, such that one cannot distinguish between a random injective function in the family and a random lossy function in the family. An injective function can be generated together with a trapdoor, which allows one to efficiently invert the function, whereas a lossy function “loses” most information about its preimage, since its image is much smaller than its domain.

### Definition 2.2 (( $T, \omega$ )-Lossy Trapdoor Family)

A quadruple  $(\text{InjGen}, \text{LossyGen}, \text{Eval}, \text{Inv})$  of PPT algorithms is said to be a  $(T, \omega)$ -lossy trapdoor function family if there exist polynomials  $n = n(\lambda)$ ,  $n' = n'(\lambda)$ ,  $s = s(\lambda)$  and  $t = t(\lambda)$  for which the following syntax and properties are satisfied:

- **Syntax.**
  - $\text{InjGen}(1^\lambda)$  takes as input a security parameter  $1^\lambda$  and outputs a pair  $(k, \text{td})$ , where  $k \in \{0, 1\}^s$  is a key corresponding to an injective function and  $\text{td} \in \{0, 1\}^t$  is a corresponding trapdoor.
  - $\text{LossyGen}(1^\lambda)$  takes as input a security parameter  $1^\lambda$  and outputs a key  $k \in \{0, 1\}^s$  corresponding to a lossy function.
  - $\text{Eval}(k, x)$  takes as input a key  $k \in \{0, 1\}^s$  and an element  $x \in \{0, 1\}^n$  and outputs an element  $y \in \{0, 1\}^{n'}$ .

- $\text{Inv}(k, \text{td}, y)$  takes as input a key  $k \in \{0, 1\}^s$ , a trapdoor  $\text{td} \in \{0, 1\}^t$ , and an element  $y \in \{0, 1\}^n$ , and outputs an element  $x \in \{0, 1\}^n \cup \{\perp\}$ .
- **Properties.** The following properties hold:
  - **Injective Mode.** For every  $\lambda \in \mathbb{N}$  and every  $k \in \text{InjGen}(1^\lambda)$  the function  $\text{Eval}(k, \cdot)$  is injective. Furthermore, for every  $x \in \{0, 1\}^{n(\lambda)}$ ,  $\Pr[\text{Inv}(k, \text{td}, \text{Eval}(k, x)) = x] = 1$ .<sup>9</sup>
  - **$\omega$ -Lossiness.** For every  $\lambda \in \mathbb{N}$  and every  $k \in \text{LossyGen}(1^\lambda)$  the function  $\text{Eval}(k, \cdot)$  has an image of size  $2^{n(\lambda) - \omega(\lambda)}$ .
  - **$T$ -Security.** There exists a negligible function  $\mu$  such that for every poly( $T$ )-size adversary  $\mathcal{A}$  and for every  $\lambda \in \mathbb{N}$ ,

$$\left| \Pr_{k \leftarrow \mathcal{G}. \text{LossyGen}(1^\lambda)} [\mathcal{A}(k) = 1] - \Pr_{k \leftarrow \mathcal{G}. \text{InjGen}(1^\lambda)} [\mathcal{A}(k) = 1] \right| = \mu(T(\lambda))$$

**Theorem 2.1** [*PW08, FGK+10*]. Assuming the sub-exponential hardness of DDH, for every constant  $0 < \delta < 1$  and every polynomial  $n(\lambda)$ , there exists a constant  $0 < \epsilon < 1$  for which there exists a  $(T, \omega)$ -lossy trapdoor function family for  $\omega(\lambda) = n(\lambda) - \lambda^\delta$  and  $T = 2^{\lambda^\epsilon}$ .

Moreover, after a  $\text{td}$ -independent preprocessing step, inversion of this function family has threshold circuits of depth  $O(d)$ , provided that large fan-in multiplication over the DDH group can be computed in depth  $d$ .

*Proof.* We slightly modify the construction due to [FGK+10] in order to obtain  $\text{TC}^0$  inversion:

- The public key is of the form  $g^M$ , where  $g$  is a generator for an order  $p$  group where DDH is hard and  $M$  is a  $k \times k$  matrix with entries in  $\{0, 1, \dots, p - 1\}$ . In *injective mode*,  $M$  is a uniformly random invertible matrix. In *lossy mode*,  $M$  is a uniformly random rank 1 matrix. Injective and lossy modes are computationally indistinguishable under the DDH assumption.
- The input  $x$  is an element of  $\{0, 1\}^n$ . To evaluate the function, one computes  $f_{g^M}(x) = g^{Mx}$  by evaluating the matrix-vector product “in the exponent.”
- The *trapdoor* in injective mode is as follows:

$$\text{td} = [a_{ij}]_{ij} = M^{-1},$$

To invert, we compute  $f_{\text{td}}^{-1}(g^z) = g^{M^{-1}z}$ , where the matrix-vector product

$$M^{-1}z = \left( \sum_j a_{ij} z_j \right)_i$$

is computed by *exponentiating*  $g^{z_j} \mapsto g^{a_{ij}z_j}$  and then computing  $k$  different  $k$ -fold products. Algorithmically, this is done as follows:

<sup>9</sup> Following [JKKZ21], we require perfect correctness for simplicity only.

- First compute  $g_{j,\ell} = g^{2^\ell z_j}$  for all  $0 \leq j \leq \log p$ . This does not require  $\text{td}$  and is thus considered a preprocessing step.
- Given  $\{g_{j,\ell}\}$ ,  $\text{td} = [a_{ij}]_{ij}$ , compute (for all  $i$ , in parallel)

$$g^{x_i} = \prod_{j,\ell} g_{j,\ell}^{a_{ij}[\ell]},$$

where  $a_{ij}[\ell]$  denotes the  $\ell$ th bit of  $a_{ij}$ .  $x_i$  can then be recovered by checking whether the group element is  $\text{id}_{\mathbb{G}}$  or  $g$ . Since this online step consists of many parallel iterated product operations, its threshold circuit depth essentially matches that of iterated group multiplication.

- Finally, we observe that in lossy mode,  $f_{g^M}$  maps a domain of size  $p^k$  to a range of size  $p$ , thus achieving the desired amount of lossiness for  $p = 2^\lambda$  and  $k = \lambda^{1/\delta}$ .

### 2.3 Correlation-Intractable Hash Families

In this section, we recall the notion of a correlation-intractable (CI) hash family originally defined in [CGH98]. We start by recalling the notion of a hash family.

**Definition 2.3.** *A hash family  $\mathcal{H}$  consists of two algorithms  $(\mathcal{H}.\text{Gen}, \mathcal{H}.\text{Hash})$ , and parameters  $n = n(\lambda)$  and  $m = m(\lambda)$ , such that:*

- $\mathcal{H}.\text{Gen}$  is a PPT algorithm that takes as input a security parameter  $1^\lambda$  and outputs a key  $k$ .
- $\mathcal{H}.\text{Hash}$  is a polynomial time computable (deterministic) algorithm that takes as input a key  $k \in \mathcal{H}.\text{Gen}(1^\lambda)$  and an element  $x \in \{0, 1\}^{n(\lambda)}$  and outputs an element  $y \in \{0, 1\}^{m(\lambda)}$ .

In what follows when we refer to a hash family, we usually do not mention the parameters  $n$  and  $m$  explicitly.

**Definition 2.4 (T-Correlation Intractable [CGH98]).** *A hash family  $\mathcal{H} = (\mathcal{H}.\text{Gen}, \mathcal{H}.\text{Hash})$  is said to be T-correlation intractable (T-CI) for a function family  $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$  if the following two properties hold:*

- For every  $\lambda \in \mathbb{N}$ , every  $f \in \mathcal{F}_\lambda$ , and every  $k \in \mathcal{H}.\text{Gen}(1^\lambda)$ , the functions  $f$  and  $\mathcal{H}.\text{Hash}(k, \cdot)$  have the same domain and the same co-domain.
- For every poly( $T$ )-size  $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$  there exists a negligible function  $\mu$  such that for every  $\lambda \in \mathbb{N}$  and every  $f \in \mathcal{F}_\lambda$ ,

$$\Pr_{\substack{k \leftarrow \mathcal{H}.\text{Gen}(1^\lambda) \\ x \leftarrow \mathcal{A}(k)}}} [\mathcal{H}.\text{Hash}(k, x) = f(x)] = \mu(T(\lambda)).$$

**Theorem 2.2. [JJ21]** *Assuming sub-exponential hardness of DDH against polynomial-time attackers, there exists a constant  $\epsilon > 0$  and a T-correlation intractable hash family for  $\text{TC}^0$ , for  $T = T(\lambda) = 2^{\lambda^\epsilon}$ .*



### 2.4 Lossy CI Hash Functions

In this section we recall the notion of a *lossy* CI hash family, originally defined in [JKKZ21].

**Definition 2.5** ( $(T, T', \omega)$ -Lossy CI). *A hash family*

$$\mathcal{H} = (\mathcal{H}.\text{Gen}, \mathcal{H}.\text{LossyGen}, \mathcal{H}.\text{Hash})$$

is said to be  $(T, T', \omega)$ -lossy CI for a function family  $\mathcal{F}$  if the following holds:

- $(\mathcal{H}.\text{Gen}, \mathcal{H}.\text{Hash})$  is a  $T$ -CI hash family for  $\mathcal{F}$  (Definition 2.4).
- The additional key generation algorithm  $\mathcal{H}.\text{LossyGen}$  takes as input a security parameter  $\lambda$  and outputs hash key  $k$ , such that the following two properties hold:
  - **$T'$ -Key Indistinguishability.** For every  $\text{poly}(T')$ -size adversary  $\mathcal{A}$ , there exists a negligible function  $\mu$  such that for every  $\lambda \in \mathbb{N}$

$$\left| \Pr_{k \leftarrow \mathcal{H}.\text{LossyGen}(1^\lambda)} [\mathcal{A}(k) = 1] - \Pr_{k \leftarrow \mathcal{H}.\text{Gen}(1^\lambda)} [\mathcal{A}(k) = 1] \right| = \mu(T'(\lambda)).$$

- **$\omega$ -Lossiness.** For every  $\lambda \in \mathbb{N}$  and every  $k \in \mathcal{H}.\text{LossyGen}(1^\lambda)$ , denoting by  $n = n(\lambda)$  the length of elements in the domain of  $\mathcal{H}.\text{Hash}(k, \cdot)$ ,

$$|\{\mathcal{H}.\text{Hash}(k, x)\}_{x \in \{0,1\}^{n(\lambda)}}| \leq 2^{n(\lambda) - \omega(\lambda)}.$$

**Theorem 2.3.** *There exists a  $(T, T', \omega)$ -lossy CI hash family for  $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$  (Definition 2.4) assuming the existence of the following primitives:*

- A  $(T', \omega)$ -lossy trapdoor function family  $\mathcal{G}$  (Definition 2.2), such that for every  $\lambda \in \mathbb{N}$ ,  $f \in \mathcal{F}_\lambda$ , and  $k \in \mathcal{G}.\text{Gen}(1^\lambda)$ , the domain of  $\mathcal{G}.\text{Eval}(k, \cdot)$  is equal to the domain of  $f$ .
- A  $T$ -CI hash family  $\mathcal{H}$  (Definition 2.4) for the function family  $\mathcal{F}'$ , where the family  $\mathcal{F}' = \{\mathcal{F}'_\lambda\}_{\lambda \in \mathbb{N}}$  is defined as follows: for each  $\lambda \in \mathbb{N}$ ,  $f' \in \mathcal{F}'_\lambda$  if and only if there exists  $f \in \mathcal{F}_\lambda$ , and  $(k, \text{td}) \in \mathcal{G}.\text{Gen}(1^\lambda)$  such that  $f'_\lambda(\cdot) = f_\lambda(\mathcal{G}.\text{Inv}(k, \text{td}, \cdot))$ . In fact, this holds even when  $\mathcal{G}.\text{Inv}(k, \text{td}, \cdot)$  is replaced by the online phase of an offline/online (with respect to  $\text{td}$ ) algorithm for  $\mathcal{G}.\text{Inv}$ .

### 2.5 SNARGs for Bounded Depth Computations

In this section we recall the main theorem from [JKKZ21], which claims that (a variant of) the GKR protocol has a standard model Fiat-Shamir instantiation. The GKR protocol considered in [JKKZ21], as well as the one considered in this work, is slightly different from the original protocol proposed in [GKR08], and we elaborate on this protocol in Sect. 5.2. In what follows, when we refer to the GKR protocol we refer to the protocol from [JKKZ21].

The GKR protocol is a publicly verifiable interactive proof for proving the correctness of log-space uniform bounded depth computations. Let  $C$  be a log-space uniform circuit of depth  $d$  and size  $s$ . The GKR protocol for proving that

$C(x) = 1$  for a given input  $x \in \{0, 1\}^n$ , consists of  $d = d(n)$  sub-protocols. Each sub-protocol is a sum-check protocol with  $\log s$  variables over a finite field  $\mathbb{F}$  of size  $\text{poly}(|C|)$ . In [GKR08] and [JKKZ21] the finite field  $\mathbb{F}$  is taken to be an extension of  $\text{GF}[2]$ . In this work we take a particular field of size  $2^\lambda$ , for which computing roots of a degree-3 univariate polynomial can be done in  $\text{TC}^0$ . See Sect. 3 for details.

In what follows, for any field ensemble  $\mathbb{F} = \{\mathbb{F}_n\}_{n \in \mathbb{N}}$  and any  $c = c_n \in \mathbb{N}$  we let  $\text{GKR}_{\mathbb{F},c}$  denote an instantiation of the GKR protocol with the field  $\mathbb{F}$  and where the degree of each variable in the underlying sum-check protocols inside the GKR protocol is bounded by  $c$ . We let  $\mathcal{F} = \mathcal{F}_{\mathbb{F},c}$  be the function family where each  $f \in \mathcal{F}$  has a degree  $c$  univariate polynomial  $p : \mathbb{F} \rightarrow \mathbb{F}$  hardwired into it. It takes as input a degree  $c$  polynomial  $p'$  specified by  $c + 1$  elements in  $\mathbb{F}$ , and it outputs a root of  $p - p'$  (which is an element in  $\mathbb{F}$ ).

**Theorem 2.4.** [JKKZ21] *Fix any field ensemble  $\mathbb{F} = \{\mathbb{F}_n\}_{n \in \mathbb{N}}$  and any  $c = c_n \in \mathbb{N}$ . Let  $\ell$  denote the number of rounds in each sum-check protocol in  $\text{GKR}_{\mathbb{F},c}$ . Fix any  $T'(\lambda) \geq \lambda$ . Assume there exists a constant  $\epsilon > 0$  for which there exists a  $(T, T', \omega)$ -lossy CI hash family for the function family  $\mathcal{F}_{\mathbb{F},c}$ , with  $T(\lambda) = 2^{\ell \cdot \lambda^\epsilon}$  and  $\omega(\lambda) = n(\lambda) - \lambda^\epsilon$ . Then there exists a hash family  $\mathcal{H}$  such that applying the Fiat-Shamir heuristic to the  $\text{GKR}_{\mathbb{F},c}$  protocol with the hash family  $\mathcal{H}$  results with a  $T'$ -sound SNARG scheme.*

In Sect. 5 we show that any log-space uniform computation has a  $\text{GKR}_{\mathbb{F},c}$  protocol with  $\mathbb{F}$  being any finite field ensemble of size  $|\mathbb{F}| = 2^\lambda$  and with  $c = 3$ . Moreover, in Sect. 3 we show that computing a root of a degree 3 univariate polynomial over a specific finite field ensemble  $\mathbb{F}$  (constructed by Healy and Viola [HV06]) can be done in  $\text{TC}^0$ . This, together with Theorem 2.4 and Theorems 2.1 and 2.2, yields our SNARG construction (Theorem 5.1).

### 3 Root-Finding in $\text{TC}^0$

In this section, we recall the finite field ensemble constructed by Healy and Viola [HV06], who show that their fields admit  $\text{TC}^0$  circuits for many basic finite field operations (addition, pairwise multiplication, large fan-in multiplication, and exponentiation). We construct explicit degree-2 extensions of all finite fields in this ensemble and prove that the same basic operations in the field extensions have  $\text{TC}^0$  circuits. Finally, we show that there are  $\text{TC}^0$  circuits finding all roots of a given quadratic or cubic equation in the original field ensemble of [HV06].

The results of this section will be used in later subsections to instantiate the Fiat-Shamir transform and show PPAD-hardness (Sect. 4) and delegation for bounded-depth computations (Sect. 5).

#### 3.1 Basic Finite Field Operations

Following [HV06], we define the field ensemble  $\{\mathbb{K}_n\}_{n=2 \cdot 3^\ell}$  as follows.

**Definition 3.1 (Healy-Viola Fields).** *The Healy-Viola (HV) field  $\mathbb{K}_n$ , which is an extension of  $\mathbb{F}_2$  of degree  $n = 2 \cdot 3^\ell$ , is defined to be the polynomial ring  $\mathbb{F}_2[x]/(x^{2 \cdot 3^\ell} + x^{3^\ell} + 1)$ .*

**Theorem 3.1 (Healy-Viola [HV06]).** *The field ensemble  $\{\mathbb{K}_n\}$  admits a polynomial-size<sup>10</sup>  $\text{TC}^0$  circuit family for the following operations:*

- Addition:  $(\alpha_1, \dots, \alpha_t) \mapsto \sum_{i=1}^t \alpha_i$  over  $\mathbb{K}$ .
- (Large fan-in) Multiplication:  $(\alpha_1, \dots, \alpha_t) \mapsto \prod_{i=1}^t \alpha_i$  over  $\mathbb{K}$ .
- Exponentiation:  $(\alpha, T) \mapsto \alpha^T$  over  $\mathbb{K}$ . The  $\text{TC}^0$  circuit size is  $\text{poly}(n, \log T)$ .

In this work, we need to extend Theorem 3.1 to hold over not just  $\mathbb{K}$  but a degree-2 extension  $\mathbb{L}/\mathbb{K}$ .

**Definition 3.2 (Degree-2 field extension of HV fields).** *The degree-2 field extension  $\{\mathbb{L}_n\}_{n=2 \cdot 3^\ell}$  of  $\mathbb{K}_n$  is defined to be the polynomial ring  $\mathbb{L} = \mathbb{K}[y]/(y^2 + y + \omega)$ , where  $\omega = x^{3^\ell} \in \mathbb{K}$ .*

We first show that the polynomial  $y^2 + y + \omega$  is irreducible over  $\mathbb{K}$  (so that  $\mathbb{L}$  is in fact a field), which follows by the following standard algebraic argument. Since the polynomial has degree 2, it suffices to show that all the roots of  $y^2 + y + \omega$  in a fixed algebraic closure  $\overline{\mathbb{K}} = \overline{\mathbb{F}_2}$  are not in  $\mathbb{K}$ . We do so by arguing that, on the one hand, any root of  $y^2 + y + \omega$  in the algebraic closure  $\overline{\mathbb{K}}$  has degree exactly 4 over  $\mathbb{F}_2$ ,<sup>11</sup> and on the other hand,  $\mathbb{K}$  does not contain any degree-4 field elements. The latter follows from the fact that  $\deg(\mathbb{K}) = 2 \cdot 3^\ell$  is not divisible by 4, so it does not contain a subfield of degree 4 over  $\mathbb{F}_2$ .<sup>12</sup>

It remains to argue that any root of  $y^2 + y + \omega$  in the algebraic closure  $\overline{\mathbb{K}}$  has degree exactly 4 over  $\mathbb{F}_2$ . This holds by the following analysis: we know that  $\omega^2 + \omega + 1 = 0$  over  $\mathbb{K}$  (but  $\omega \notin \mathbb{F}_2$ ), so  $\mathbb{F}_2[\omega]$  has degree 2 over  $\mathbb{F}_2$ . Moreover,  $y^2 + y + \omega$  is irreducible over  $\mathbb{F}_2[\omega] \simeq \mathbb{F}_4$ . Thus, any root of  $y^2 + y + \omega$  lies in  $\mathbb{F}_{16}$  (realized as a degree 2 extension of  $\mathbb{F}_2[\omega]$ ) but not  $\mathbb{F}_4$ .

Having established that  $\mathbb{L}$  is well-defined, we proceed to generalize Theorem 3.1.

**Theorem 3.2.** *The field ensemble  $\{\mathbb{L}_n\}$  admits a polynomial-size  $\text{TC}^0$  circuit family for the following operations:*

- Addition:  $(\alpha_1, \dots, \alpha_t) \mapsto \sum_{i=1}^t \alpha_i$  over  $\mathbb{L}$ .
- (Large fan-in) Multiplication:  $(\alpha_1, \dots, \alpha_t) \mapsto \prod_{i=1}^t \alpha_i$  over  $\mathbb{L}$ .
- Exponentiation:  $(\alpha, T) \mapsto \alpha^T$  over  $\mathbb{L}$ . The  $\text{TC}^0$  circuit size is  $\text{poly}(n, \log T)$ .

Theorem 3.2 follows by a very similar approach as the proof of Theorem 3.1, making use of some additional properties of  $\mathbb{L}$ .

<sup>10</sup> As usual, the circuit size will be polynomial in the description length of its input, which will be at least  $n$  as a single field element is an  $n$ -bit string.

<sup>11</sup> An element  $\alpha$  in a field extension  $\mathbb{K}$  of  $\mathbb{F}_2$  is said to have degree  $d$  if  $d$  is the minimal degree of a nonzero polynomial  $p$  over  $\mathbb{F}_2$  such that  $p(\alpha) = 0$  (over  $\mathbb{K}$ ).

<sup>12</sup>  $\mathbb{F}_{p^d}$  is a subfield of  $\mathbb{F}_{p^n}$  if and only if  $d \mid n$ .

*Proof.* Note that  $\alpha \in \mathbb{L}$  is given as an explicit bivariate polynomial  $\alpha_0(x) + \alpha_1(x)y$  for  $\alpha_0(x), \alpha_1(x) \in \mathbb{K}$ . An  $\text{AC}^0[\oplus] \subseteq \text{TC}^0$  circuit family for addition then follows immediately by component-wise addition. Additionally, note that since

$$\begin{aligned} & \left(\alpha_0(x) + \alpha_1(x)y\right) \left(\beta_0(x) + \beta_0(x)y\right) \\ &= \alpha_0(x)\beta_0(x) + \left(\alpha_1(x)\beta_0(x) + \alpha_0(x)\beta_1(x)\right)y + \alpha_1(x)\beta_1(x)y^2 \\ &= \alpha_0(x)\beta_0(x) + \left(\alpha_1(x)\beta_0(x) + \alpha_0(x)\beta_1(x)\right)y + \alpha_1(x)\beta_1(x)(y + \omega) \\ &= \left(\alpha_0(x)\beta_0(x) + \alpha_1(x)\beta_1(x)\omega\right) + \left(\alpha_1(x)\beta_0(x) + \alpha_0(x)\beta_1(x) + \alpha_1(x)\beta_1(x)\right)y, \end{aligned}$$

an  $\text{AC}^0[\oplus] \subseteq \text{TC}^0$  circuit for pairwise multiplication over  $\mathbb{L}$  follows from the analogous circuits over  $\mathbb{K}$ .

Next, we consider large fan-in multiplication. Suppose we are given  $t$  field elements  $\alpha^{(1)}, \dots, \alpha^{(t)} \in \mathbb{L}_n$  and we want to compute  $\prod \alpha^{(i)} \in \mathbb{L}_n$ . To do this, we view each  $\alpha^{(i)}$  as a bivariate polynomial over  $\mathbb{Z}$ , and compute (in  $\text{TC}^0$ ) the bivariate polynomial representation of  $\prod \alpha^{(i)}$ . [HAB02] argues that the analogous product for *univariate* polynomials can be done in (uniform)  $\text{TC}^0$ , but we can see the same holds for our bivariate polynomials via the following reduction:

- Given a bivariate polynomial  $\alpha^{(i)}(x, y)$ , define the polynomial  $\beta^{(i)}(z) = \alpha^{(i)}(z, z^{n-t})$ ; the coefficients of  $\beta^{(i)}$  can be computed with a  $\text{TC}^0$  circuit.
- Compute the polynomial  $\prod \beta^{(i)} \in \mathbb{Z}[z]$  by invoking [HAB02].
- Map the coefficients of  $\prod \beta^{(i)}$  to the coefficients of  $\prod \alpha^{(i)}(x, y)$  via the correspondence  $z^k \mapsto x^{k \pmod{nt}} \cdot y^{\lfloor k/nt \rfloor}$ ; this map can also be computed in  $\text{TC}^0$ .

Finally, we must reduce this bivariate polynomial  $\prod \alpha^{(i)}(x, y)$  modulo  $(x^{2 \cdot 3^\ell} + x^{3^\ell} + 1, y^2 + y + x^{3^\ell})$ ; this can be done via the following process:

- Reduce each  $y$  exponent modulo 15 (since  $y^{15} \equiv 1$ , as  $y \in \mathbb{L}$  is in a degree 4 extension of  $\mathbb{F}_2$ ),
- Reduce each (constant) power of  $y$  modulo  $(y^2 + y + x^{3^\ell}, x^{2 \cdot 3^\ell} + x^{3^\ell} + 1)$ ,
- Group terms by power of  $y$  (either  $y^0$  or  $y^1$ ), and
- Reduce each  $y^j$  coefficient modulo  $x^{2 \cdot 3^\ell} + x^{3^\ell} + 1$ .

This completes the proof that large fan-in multiplication over  $\mathbb{L}$  is in  $\text{TC}^0$ .

Finally, we consider exponentiation  $(\alpha, T) \mapsto \alpha^T \in \mathbb{L}$ .  $T$  is given as input in binary; by invoking a large fan-in multiplication solver, we can reduce to the case where  $T = 2^i$  is a power of 2. Now, note that in  $\mathbb{L}$ , we have

$$\alpha(x, y)^{2^i} = \alpha\left(x^{2^i}, y^{2^i}\right) = \alpha\left(x^{2^i}, y + \sum_{j=0}^{i-1} \omega^{2^j}\right),$$

where the first equality follows from the fact that our field has characteristic 2, and the second equality uses the defining equation  $y^2 + y + \omega = 0$ . The field element  $g(\omega) = \sum_{j=0}^{i-1} \omega^{2^j} \in \mathbb{K}$  can be computed in  $\text{AC}^0[\oplus]$  (e.g. invoking [HV06]),

and  $\alpha(\cdot, \cdot)$  is linear in its second argument, so we can compute  $\alpha^{2^i} \in \mathbb{L}$  by computing each expression  $x^{2^i \cdot k}$  for  $k \leq n$ , which by [HV06] can be done in  $\text{AC}^0[\oplus]$ , and invoking pairwise field element multiplication and large fan-in addition circuits.

### 3.2 Finding Roots of $\mathbb{K}$ -quadratics in $\mathbb{L}$

In this section, we give a  $\text{TC}^0$  circuit family for solving the following computational problem:

**Definition 3.3** ( $(\mathbb{K}, \mathbb{L})$  Quadratic Root Finding). *Given a quadratic polynomial  $az^2 + bz + c \in \mathbb{K}[z]$ , find all zeroes of this polynomial in  $\mathbb{L}$ .*

**Theorem 3.3.**  $(\mathbb{K}, \mathbb{L})$  quadratic root finding admits a  $\text{TC}^0$  circuit family.

*Proof.* We break into cases.

- If  $a = 0$ , then this amounts to computing  $b^{-1} \in \mathbb{K}$ , which can be done because  $b^{-1} = b^{2^n - 2}$  and exponentiation is in  $\text{TC}^0$  (Theorem 3.1).
- If  $a \neq 0$  and  $b = 0$ , then this amounts to inverting  $a$  and computing a square root in  $\mathbb{K}$ , which can be done because  $\sqrt{\alpha} = \alpha^{2^{n-1}}$  for  $\alpha \in \mathbb{K}$ .
- If  $a \neq 0$  and  $b \neq 0$ , then (by invoking standard field operations) this reduces to the case where  $a = 1$  and  $b = 1$ , as

$$az^2 + bz + c = 0 \iff (a/b \cdot z)^2 + (a/b \cdot z) + a/b^2 \cdot c = 0.$$

Thus, for the rest of the proof, we assume that  $a = 1$  and  $b = 1$ . Moreover, it suffices to find a single solution  $z^*$  in  $\mathbb{L}$ , as the other solution will be  $z^* + 1$  (since  $\mathbb{L}$  has characteristic 2).

Given  $z^2 + z + c = 0$ , since  $n = 2 \cdot 3^\ell$  is 2 mod 4, solving the equation turns out (via standard theory of finite fields, see e.g. [BSS99] Chapter II) to be related to the  $\mathbb{F}_4$ -trace map

$$\text{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha) = \sum_{i=0}^{n/2-1} \alpha^{2^{2i}}$$

as follows. First, we note that for any  $\alpha \in \mathbb{K}$ ,  $\text{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha) \in \mathbb{F}_2[\omega]$ , as  $\text{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha)$  is invariant under the map  $z \mapsto z^{2^i}$  for all even  $i$ . Additionally, the formula above is computable via a  $\text{TC}^0$  (in fact,  $\text{AC}^0[\oplus]$ ) circuit family.

Next, we give a  $\text{TC}^0$  (in fact,  $\text{AC}^0[\oplus]$ ) circuit that on input  $\alpha \in \mathbb{K}$ , outputs  $\beta \in \mathbb{K}$  such that  $\beta^2 + \beta = \alpha + \text{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha)$ . The circuit simply computes the expression

$$\beta = \sum_{\substack{0 \leq i \leq n/2-1 \\ i \text{ odd}}} \alpha^{2^{2i}} + \alpha^{2^{2i+1}}.$$

Observe that

$$\beta^2 + \beta = \sum_{\substack{0 \leq i \leq n/2-1 \\ i \text{ odd}}} \alpha^{2^{2i}} + \alpha^{2^{2i+2}} = \alpha + \text{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha),$$

where the last equation uses the fact that  $n/2 - 1$  is even.

Finally, in order to solve the equation  $z^2 + z + \alpha = 0$ , given that we can compute  $\beta$  above, it suffices by additivity to be able to solve the equation  $z^2 + z + c = 0$  for  $c = \text{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha) \in \mathbb{F}_2[\omega]$ . But this can be done by lookup table: for  $c = 0$  a solution is 0, for  $c = 1$  a solution is  $\omega$ , for  $c = \omega$  a solution is  $y$ , and for  $c = 1 + \omega$  a solution is  $\omega + y$ . This completes the proof of Theorem 3.3.

### 3.3 Finding Roots of Cubics in $\mathbb{K}$

In this section, we give a  $\text{TC}^0$  circuit family for solving the following computational problem:

**Definition 3.4** ( $(\mathbb{K}, \mathbb{K})$  Cubic Root Finding). *Given a cubic polynomial  $az^3 + bz^2 + cz + d \in \mathbb{K}[z]$ , find all zeroes of this polynomial that lie in  $\mathbb{K}$ .*

**Theorem 3.4.**  *$(\mathbb{K}, \mathbb{K})$ -cubic root finding admits a  $\text{TC}^0$  circuit family.*

*Proof.* If  $a = 0$ , then by Theorem 3.3, we know that  $(\mathbb{K}, \mathbb{L})$ -quadratic root finding can be solved in  $\text{TC}^0$ , and it is easy to check membership in  $\mathbb{K}$  (on an input  $\alpha \in \mathbb{L}$ ), so this suffices to solve  $(\mathbb{K}, \mathbb{K})$ -quadratic root finding as well.

Thus, we now assume that  $a = 1$ . Note that we only want to find all roots in  $\mathbb{K}$ , so we may assume without loss of generality that **there is at least one root in  $\mathbb{K}$**  (or else the problem is vacuous). Under this promise, it follows that *all three roots* will lie in  $\mathbb{L}$  (since the polynomial factors into linear and quadratic terms over  $\mathbb{K}$ ). Our algorithm will find all three of these roots (and then check membership in  $\mathbb{K}$ ).

We find these roots by invoking (a special case of) a standard characteristic 2 variant of the cubic formula (following e.g. [Lag70]). Namely, letting  $\alpha_0, \alpha_1, \alpha_2$  denote the three roots in  $\mathbb{L}$ , we will find  $\alpha_0, \alpha_1, \alpha_2$  by first solving a related quadratic equation with coefficients in  $\mathbb{K}$ , then taking cube roots (in  $\mathbb{L}$ ), and then solving a linear system over  $\mathbb{L}$ .

By Vieta’s identities, we know that  $\hat{\alpha}_0 := \alpha_0 + \alpha_1 + \alpha_2 = b$ . Letting  $\omega = x^{3^\ell} \in \mathbb{K}$  so that  $\omega^3 = 1$ , we will eventually also compute the linear combinations

$$\hat{\alpha}_1 = \alpha_0 + \omega\alpha_1 + \omega^2\alpha_2,$$

$$\hat{\alpha}_2 = \alpha_0 + \omega^2\alpha_1 + \omega\alpha_2$$

The map  $(\alpha_0, \alpha_1, \alpha_2) \mapsto (\hat{\alpha}_0, \hat{\alpha}_1, \hat{\alpha}_2)$  is always (efficiently) invertible over  $\mathbb{L}$ , so it suffices to compute  $\hat{\alpha}_1, \hat{\alpha}_2$ . This is sometimes referred to as the “Lagrange resolvent method.”

The field elements  $\hat{\alpha}_1$  and  $\hat{\alpha}_2$  have been carefully chosen to satisfy useful symmetries when  $\alpha_0, \alpha_1, \alpha_2$  are permuted as formal variables:

- Under the cyclic permutation  $(\alpha_0, \alpha_1, \alpha_2) \mapsto (\alpha_i, \alpha_{i+1}, \alpha_{i+2})$ , we have that  $\hat{\alpha}_1 \mapsto \omega^i \hat{\alpha}_1$  and  $\hat{\alpha}_2 \mapsto \omega^{2i} \hat{\alpha}_2$ .
- Under the swap permutation  $\alpha_i \leftrightarrow \alpha_j$ , we have that  $\hat{\alpha}_1 \mapsto \omega^{i+j} \hat{\alpha}_2$  and  $\hat{\alpha}_2 \mapsto \omega^{2i+2j} \hat{\alpha}_1$ .

The symmetries simplify even further if you consider  $\hat{\alpha}_1^3$  and  $\hat{\alpha}_2^3$  (since  $\omega^3 = 1$ ): under cyclic permutation, these expressions are *invariant*, while under a swap permutation, they swap!

Thus,  $\hat{\alpha}_1^3 + \hat{\alpha}_2^3$  and  $\hat{\alpha}_1^3 \hat{\alpha}_2^3$  are symmetric under all permutations of  $(\alpha_0, \alpha_1, \alpha_2)$ . The theory of symmetric polynomials therefore tells us that  $\hat{\alpha}_1^3 + \hat{\alpha}_2^3$  and  $\hat{\alpha}_1^3 \hat{\alpha}_2^3$  can be expressed in terms of the *elementary* symmetric polynomials in  $\alpha_0, \alpha_1, \alpha_2$ , which in our case evaluate to none other than  $b, c$ , and  $d$  by Vieta's identities. Indeed, one can explicitly check that

$$(\hat{\alpha}_1 \hat{\alpha}_2)^3 = (b^2 + c)^3$$

and

$$\hat{\alpha}_1^3 + \hat{\alpha}_2^3 = bc + d,$$

and thus  $(\hat{\alpha}_1^3, \hat{\alpha}_2^3)$  are solutions to the quadratic equation

$$z^2 + (bc + d)z + (b^2 + c)^3 = 0.$$

By Theorem 3.3, we can hence compute  $\hat{\alpha}_1^3, \hat{\alpha}_2^3 \in \mathbb{L}$  with a  $\text{TC}^0$  circuit. Finally, since  $\hat{\alpha}_1, \hat{\alpha}_2 \in \mathbb{L}$ , we can find three candidate values for each of  $\hat{\alpha}_1, \hat{\alpha}_2$ , by computing cube roots over  $\mathbb{L}$ ; this leads to nine possible root sets for our original problem, which can then be individually checked to find the correct roots.

Thus, we have reduced the problem to computing cube roots over  $\mathbb{L}$ . For this problem, we use a special case of the Adleman-Manders-Miller algorithm [AMM77]. Specifically, we note that  $|\mathbb{L}| - 1 = 2^{4 \cdot 3^\ell} - 1$  is congruent to  $3^{\ell+1}$  modulo  $3^{\ell+2}$ . Then, invoking exponentiation<sup>13</sup> in  $\mathbb{L}$ , on any input  $\alpha \in \mathbb{L}$  we can compute

$$\beta = \alpha^{\frac{|\mathbb{L}| - 1 - 3^{\ell+1}}{3^{\ell+2}}} \in \mathbb{L}.$$

Note that

$$\beta^{3^{\ell+2}} = \alpha^{3^{\ell+1}},$$

and thus  $\beta^3/\alpha$  is a  $3^{\ell+1}$ th root of unity, the set of which is precisely  $S = \{1, x, x^2, \dots, x^{3^{\ell+1}-1}\}$ . We can then enumerate (in parallel) over this  $\leq n$ -size set to determine (the  $x$ -exponent of)  $\beta^3/\alpha$  and thus compute a cube root of  $\alpha$  provided that a cube root of  $\beta^3/\alpha$  exists (necessarily within  $S$ ).

Putting everything together, we obtain the desired  $\text{TC}^0$  circuit family for  $(\mathbb{K}, \mathbb{K})$ -cubic root finding.

## 4 PPAD-Hardness from Subexponential DDH

In this section, we prove Theorem 1.2, that PPAD is hard under the sub-exponential DDH assumption. We do this by instantiating the Fiat-Shamir heuristic for the sumcheck protocol executed on polynomials of individual degree

<sup>13</sup> The (large) exponent can also be computed in  $\text{TC}^0$  [HAB02], or can be nonuniformly hard-wired for simplicity.

2 over the Healy-Viola field ensemble. We prove that Fiat-Shamir for this protocol is sound under DDH by using a lossy CI hash family for  $TC^0$  (Theorem 2.3) and appealing to  $TC^0$  algorithms for quadratic root finding (Theorem 3.3).

**Definition 4.1 ( $\oplus 3SAT$ ).** *A 3CNF formula  $\phi$  is in the language  $\oplus 3SAT$  if the number of satisfying assignments to  $\phi$  is odd.*

**Fact 1 ([VV85]).** *If NP is hard (on average) for PPT algorithms, then  $\oplus 3SAT$  is hard (on average) for PPT algorithms.*

In particular, if one-way functions exist, then  $\oplus 3SAT$  is hard on average.

**Definition 4.2 (Sumcheck Language).** *An instance of the sumcheck language consists of an arithmetic circuit  $f$  over some field  $\mathbb{F}$ , along with a target value  $y$ . The pair  $(f, y)$  is a YES-instance if*

$$\sum_{x \in \{0,1\}^n} f(x_1, \dots, x_n) = y.$$

In this work, we observe that if  $\oplus 3SAT$  is hard on average, then there is a hard sumcheck problem over  $\mathbb{F}_2$  where the individual degree of  $f$  is at most two.

**Lemma 4.1.** *If  $\oplus 3SAT$  is hard-on-average, then sumcheck over  $\mathbb{F}_2$  is hard-on-average with respect to a distribution of  $(f, y)$  such that the individual degree of  $f$  is at most two.*

*Proof.* We describe a one-to-one reduction mapping  $\oplus 3SAT$  formulas  $\phi$  to sumcheck polynomials  $f$ , so that deciding whether  $\phi \in \oplus 3SAT$  maps to checking whether  $(f, 1)$  is a valid sumcheck instance.

Suppose that  $\phi$  is an  $n$ -variable,  $m$ -clause 3CNF:

$$\phi(x_1, \dots, x_n) = \bigwedge_{j=1}^m \phi_j(x_{j_1}, x_{j_2}, x_{j_3})$$

where each  $\phi_j$  is an OR of three variables  $(x_{j_1}, x_{j_2}, x_{j_3})$  with some negation pattern (contained in the description of  $\phi_j$ ). Then, consider the following formula in  $3m$  variables:

$$\begin{aligned} f(z = (z_{j,k})_{j \in [m], k \in \{1,2,3\}}) \\ = \prod_{j=1}^m \phi_j(z_{j,1}, z_{j,2}, z_{j,3}) \prod_{i=1}^n \left( \prod_{j,k:j_k=i} z_{j,k} + \prod_{j,k:j_k=i} (1 - z_{j,k}) \right), \end{aligned}$$

where  $\phi_j$  can be interpreted as a multilinear polynomial in three variables over  $\mathbb{F}_2$ . We observe that:

- $f$  has individual degree at most 2. This is because the two products are individually multilinear.



- For  $z \in \{0, 1\}^{3m}$ ,  $f(z) = 1$  if and only if for some  $x \in \{0, 1\}^n$ ,  $\phi(x) = 1$  and  $z_{j,k} = x_{j_k}$  for all  $(j, k)$ . Otherwise,  $f(z) = 0$ .

Thus, we see that

$$\sum_{x \in \{0,1\}^n} \phi(x_1, \dots, x_n) = \sum_{y \in \{0,1\}^{3m}} f(y) \pmod{2}.$$

This completes the reduction.

To conclude that PPAD is hard-on-average, we combine Lemma 4.1 with the unambiguous non-interactive argument system for sumcheck from [JKKZ21]. [JKKZ21] implies the following result:

**Theorem 4.1 ([JKKZ21], translated).** *Let  $K$  be a field (ensemble) of size  $2^\lambda$ . Then, there exists an updatable, unambiguous non-interactive argument system for  $\text{Sumcheck}_K$  for individual degree  $d$  polynomials assuming the existence of a hash family  $\mathcal{H}$  that is lossy CI (Definition 2.5) for a class of functions that enumerate over all roots of a given univariate degree  $d$  polynomial over  $K$ .*

By Theorem 2.2, Theorem 2.3, and Theorem 2.1, we know that there exists a lossy CI hash family for  $TC^0$  circuits under sub-exponential DDH. Moreover, letting  $\{K_\lambda\}$  denote the field ensemble defined in Definition 3.1, we showed that roots of degree 2 polynomials over  $K$  can be enumerated in  $TC^0$ . Thus, by Theorem 4.1, we conclude that the claimed argument system exists under sub-exponential DDH.

Finally, it is known that an argument system satisfying the conditions of Theorem 4.1 (along with the hardness of the underlying sumcheck problem) implies the hardness of PPAD [CHK+19a], so this completes the proof of Theorem 1.2.

## 5 Delegation for Bounded Depth Computations from Subexponential DDH

In this section, we apply and extend our techniques to prove our main theorem on SNARGs for bounded-depth computation.

**Theorem 5.1.** *Assuming the sub-exponential hardness of the DDH assumption, there exists a SNARG for any logspace uniform depth  $d$  and size  $s$  computation, where the size of the SNARG and the crs is bounded by  $d \cdot \text{poly}(\lambda, \log s)$  and the verification time is  $(n + d) \cdot \text{poly}(\lambda, \log s)$ , where  $n$  is the length of the input.*

Our SNARG is obtained by applying the Fiat-Shamir heuristic to a variant of the GKR protocol, considered in [KPY18, JKKZ21] (building on a simplification of the original GKR protocol due to [Gol18]).

### 5.1 Variable-Extended Formulations for Boolean Functions

In this section we show how to reduce the degree of any boolean formula down to individual degree at most 2, by adding auxiliary variables. Loosely speaking, this is done by adding a variable corresponding to each wire in the original formula, and computing the original formula by making a series of consistency checks.

**Definition 5.1.** *Let  $f(x_1, \dots, x_m)$  be a boolean function on  $m$  variables. We say that  $g(x_1, \dots, x_m, z_1, \dots, z_t)$  is a variable-extended formulation of  $f$  if for every  $x \in \{0, 1\}^m$ , there exists a unique  $z(x) \in \{0, 1\}^t$  such that  $g(x, z(x)) = f(x)$ , and  $g(x, z) = 0$  for all  $z \neq z(x)$ .*

**Lemma 5.1.** *Let  $f(x_1, \dots, x_m)$  be a NAND-boolean formula of size  $s$ . Then, there exists a variable-extended formulation  $g$  of  $f$  such that (1)  $t = s$ , and (2)  $g$  can be computed by a  $\mathbb{F}_2$ -arithmetic circuit of size  $O(s)$  that defines a (formal) polynomial of individual degree at most 2.*

*Also, the above arithmetic circuit can be constructed in time  $\text{poly}(s)$  given the description of  $f$ .*

*Proof.* We use a similar strategy as in Lemma 4.1. That is, we introduce  $s$  new variables  $z_1, \dots, z_s$ , one for each wire of the formula computing  $f$ . We then define

$$g(x, z) = z_s \prod_{i=1}^s g_i(z) \prod_{j=1}^m g'_j(x, z),$$

where for every gate  $(i, j, k)$  we have  $g_i(z) = z_i + z_j z_k$  and for every input index  $j$  we have  $g'_j(x, z) = x_j \prod_{i \in S_j} z_i + (1 - x_j) \prod_{i \in S_j} (1 - z_i)$ , where  $S_j$  denotes the set of leaf indices corresponding to  $x_j$ . Note that  $g(x, z)$  has individual degree 2, since (1)  $z_s$  appears only twice, (2) each intermediate (non-output, non-leaf) variable only appears twice because they have fan-in 1 and fan-out 1, and (3) the variables  $\{x_j, z_i\}_{j \in [m], i \in S_j}$  have degree at most 2 (they occur at most once in the first product, while the second product is multilinear).

### 5.2 A GKR Protocol with Degree 3 Sumcheck Polynomials

In this section, we construct a special variant of the GKR interactive proof system for logspace-uniform depth- $d$  computation. Our starting point is the GKR protocol variant described in [KPY18, JKKZ21], which makes use of observations from [Mei13, Gol18] to simplify the protocol. In [JKKZ21], it was shown that the Fiat-Shamir heuristic can be instantiated for this protocol using a hash function that is “lossy correlation-intractable” for circuits that (modulo basic field operations) compute roots of univariate polynomials of polylogarithmic degree. They then show how to construct such a lossy correlation-intractable hash functions from the sub-exponential LWE assumption.

By using an appropriate *variable-extended formulation* (Lemma 5.1), we will modify the protocol so that every sumcheck sub-protocol uses a polynomial of individual degree *at most* 3. Finally, working over the field ensemble from Definition 3.1 and using the correlation-intractable hash family of [JJ21] (and lossy trapdoor functions from DDH [PW08]), we will deduce Theorem 5.1.

*The Protocol.* Let  $C = \{C_n\}_n$  denote a family of logspace-uniform circuits of depth  $d$  and width  $w$ . We assume without loss of generality that  $C$  has fan-in 2 and consists of addition (mod 2) and multiplication (mod 2) gates. The key objects of interest are the *gate-indicator functions*  $\chi_{\text{add}}^{(i)}, \chi_{\text{mult}}^{(i)}$  for each layer  $(i)$  of the circuit.  $\chi_{\text{add}}^{(i)}$  and  $\chi_{\text{mult}}^{(i)}$  take as input three strings  $(a, b, c) \in \{0, 1\}^{\log w}$  and output whether  $(a, b, c)$  is an addition (respectively, multiplication) gate in  $C$ .

The protocol is typically defined with respect to particular *low-degree extensions*  $\tilde{\chi}_{\text{add}}^{(i)}, \tilde{\chi}_{\text{mult}}^{(i)}$  of  $\chi_{\text{add}}, \chi_{\text{mult}}$ . For our variant, we make use of the following fact shown implicitly in [Gol18]:

**Fact 2.** *Let  $C'$  be any family of logspace-uniform circuits of depth  $d$  and size  $s$ . Then, there exists a family  $C$  of logspace-uniform circuits of depth  $d \cdot \text{poly} \log(s)$  and size  $\text{poly}(s)$  such that:*

- $C$  computes the same function as  $C'$ , and
- For all  $i$ ,  $\chi_{\text{add}}^{(i)}, \chi_{\text{mult}}^{(i)}$  (for  $C$ ) are computable by boolean formulas of size  $O(\log w)$  (i.e., the size is linear in the  $\chi_{\text{add}}, \chi_{\text{mult}}$  input length). These formulas can be constructed (by a uniform Turing machine) in time  $\text{poly}(\log s)$ .

[Gol18] only explicitly claims that the formulas have size  $\text{poly} \log s$ , but the construction in [Gol18] Sect. 3.4.2 actually (specializing to  $H = \{0, 1\}$ ) implies Fact 2.

Thus, we assume without loss of generality that  $C$  satisfies the conclusion of Fact 2. Invoking Lemma 5.1, we conclude that  $\chi_{\text{add}}^{(i)}, \chi_{\text{mult}}^{(i)}$  have *variable-extended formulations*  $\psi_{\text{add}}^{(i)}, \psi_{\text{mult}}^{(i)} : \{0, 1\}^{3 \log w + O(\log s)} \rightarrow \{0, 1\}$  that are computable by  $\mathbb{F}_2$ -arithmetic circuits of size  $O(\log s)$  that define polynomials of individual degree at most 2. We let  $\tilde{\psi}_{\text{add}}^{(i)}, \tilde{\psi}_{\text{mult}}^{(i)}$  denote the corresponding individual degree 2 polynomials.

We are finally ready to describe the protocol, which will use arithmetic over an extension  $K$  of  $\mathbb{F}_2$ . Our instantiation will use the field ensemble from Definition 3.1.

- The prover and verifier, given the logspace-uniform Turing machine that constructs  $C$ , both compute arithmetic circuit descriptions of each  $\tilde{\psi}_{\text{add}}^{(i)}, \tilde{\psi}_{\text{mult}}^{(i)}$ .
- The prover, given the input  $x$  and circuit  $C$ , computes the following quantities:
  - For every layer  $i$  of the circuit, compute the string  $L_i = L_i(C, x) \in \{0, 1\}^w$  consisting of all wire values in the evaluation  $C(x)$  in the  $i$ th layer of  $C$ .
  - For each such  $i$ , define the function  $\ell_i : \{0, 1\}^{\log w} \rightarrow \{0, 1\}$  such that  $\ell_i(a) = (L_i)_a$ , where  $a$  is interpreted as an integer between 0 and  $w - 1$ . Implicitly, this defines a *multi-linear extension*  $\hat{\ell}_i : K^{\log w} \rightarrow K$  of  $\ell_i$ .
- The prover and verifier recursively agree on a *pair* of claims of the form “ $\hat{\ell}_i(u_1) = v_1$ ,” “ $\hat{\ell}_i(u_2) = v_2$ ” for  $u_1, u_2 \in K^{\log w}$  and  $v_1, v_2 \in K$ . They do so as follows:
  - The base case is  $i = d$ , the top (output) layer of  $C$ ; the claims are (both) that  $\hat{\ell}_d(0^{\log w}) = y$  (where allegedly  $C(x) = y$ ).

- Inductively, suppose that we have two claims “ $\hat{\ell}_i(u_1) = v_1$ ,” “ $\hat{\ell}_i(u_2) = v_2$ ” about layer  $i$ . The recursion uses the fact that

$$\hat{\ell}_i(u) = \sum_{a \in \{0,1\}^{\log w}} \widehat{EQ}(u, a)\ell_i(a),$$

which can be written as

$$\sum_{a,b,c} \widehat{EQ}(u, a) \left( \chi_{\text{add}}^{(i)}(a, b, c) \cdot (\ell_{i-1}(b) + \ell_{i-1}(c)) + \chi_{\text{mult}}^{(i)}(a, b, c)\ell_{i-1}(b) \cdot \ell_{i-1}(c) \right),$$

which is equal to

$$\sum_{\substack{a,b,c \\ z \in \{0,1\}^t}} \widehat{EQ}(u, a) \left( \psi_{\text{add}}^{(i)}(a, b, c, z) \cdot (\ell_{i-1}(b) + \ell_{i-1}(c)) + \psi_{\text{mult}}^{(i)}(a, b, c, z)\ell_{i-1}(b) \cdot \ell_{i-1}(c) \right),$$

where  $\widehat{EQ}(u, a) := \prod_j (1 + u_j + a_j)$ .

- The prover and verifier then run two simultaneous sumcheck protocols using the polynomials  $g_{u_1}, g_{u_2}$ , where

$$g_u(a, b, c, z) = \widehat{EQ}(u, a) \cdot \left( \tilde{\psi}_{\text{add}}^{(i)}(a, b, c, z) \cdot (\hat{\ell}_{i-1}(b) + \hat{\ell}_{i-1}(c)) + \tilde{\psi}_{\text{mult}}^{(i)}(a, b, c, z)\hat{\ell}_{i-1}(b) \cdot \hat{\ell}_{i-1}(c) \right)$$

and the claimed outputs  $v_1, v_2$ . Importantly, the same verifier randomness is used for these two sumcheck protocols.

- At the end of the interactive phase of this protocol, the verifier has a tuple of field elements  $\beta \in K^{3 \log w + O(\log s)}$  and outputs  $\gamma_1, \gamma_2$  such that (allegedly)  $g_{u_1}(\beta) = \gamma_1$  and  $g_{u_2}(\beta) = \gamma_2$ . Let  $u'_1, u'_2$  denote the part of  $\beta$  corresponding to  $b$  and  $c$ .
  - Finally, the prover sends  $v'_1 = \hat{\ell}_i(u'_1), v'_2 = \hat{\ell}_i(u'_2)$  to the verifier. Since  $\widehat{EQ}, \tilde{\psi}_{\text{add}}^{(i)}, \tilde{\psi}_{\text{mult}}^{(i)}$  are all computable in time  $\text{poly}(\log s)$ , the verifier can check that  $v'_1$  and  $v'_2$  are consistent with the claims output by the sumcheck protocol. This completes the recursive step, which has produced two new claims  $(u'_1, v'_1), (u'_2, v'_2)$ .
- After this recursive process, the verifier has obtained two final claims “ $\hat{\ell}_0(u_1) = v_1$ ,” “ $\hat{\ell}_0(u_2) = v_2$ ” about the multilinear extension  $\hat{\ell}_0$ . Since  $\hat{\ell}_0$  is nothing more than the multilinear extension of the input  $x$  (thought of as a function mapping  $\{0, 1\}^{\log n} \rightarrow \{0, 1\}$ ), the verifier can check these two claims (given  $x$ ) using  $O(n)$  field operations.

Crucially,  $\tilde{\psi}_{\text{add}}$  and  $\tilde{\psi}_{\text{mult}}$  have individual degree 2, which implies that every polynomial  $g_u$  has individual degree **at most** 3. This is because  $\widehat{EQ}(u, a)(\ell_{i-1}(b) + \ell_{i-1}(c))$  and  $\widehat{EQ}(u, a)\ell_{i-1}(b)\ell_{i-1}(c)$  are both multilinear polynomials.

This completes our description of our variant of the [GKR08] protocol. By combining Theorems 2.1, 2.2 and 2.4, the fact that this [GKR08] variant runs (pairs of) degree 3 sumchecks, and Theorem 3.4, we conclude Theorem 5.1.

**Acknowledgements.** VV was supported in part by DARPA under Agreement No. HR00112020023, NSF CNS-2154174, and a Thornton Family Faculty Research Innovation Fellowship from MIT. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA. This research was conducted in part while AL was at MIT, where he was supported by a Charles M. Vest fellowship and the grants above.

## References

- [AMM77] Adleman, L., Manders, K., Miller, G.: On taking roots in finite fields. In: 18th Annual Symposium on Foundations of Computer Science (SFCS 1977), pp. 175–178. IEEE Computer Society (1977)
- [Bar01] Barak, B.: How to go beyond the black-box simulation barrier. In: 42nd FOCS, pp. 106–115. IEEE Computer Society Press, October 2001
- [BBH+19] Bartusek, J., Bronfman, L., Holmgren, J., Ma, F., Rothblum, R.D.: On the (in)security of Kilian-based SNARGs. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 522–551. Springer, Heidelberg (2019). [https://doi.org/10.1007/978-3-030-36033-7\\_20](https://doi.org/10.1007/978-3-030-36033-7_20)
- [BCH+22] Bitansky, N., et al.: PPAD is as hard as iterated squaring and LWE. In: TCC 2022 (2022). <https://eprint.iacr.org/2022/1272>
- [Ber70] Berlekamp, E.R.: Factoring polynomials over large finite fields. *Math. Comput.* **24**(111), 713–735 (1970)
- [BKM20] Brakerski, Z., Koppula, V., Mour, T.: NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 738–767. Springer, Heidelberg (2020). [https://doi.org/10.1007/978-3-030-56877-1\\_26](https://doi.org/10.1007/978-3-030-56877-1_26)
- [BPR15] Bitansky, N., Paneth, O., Rosen, A.: On the cryptographic hardness of finding a Nash equilibrium. In: Guruswami, V. (ed.) 56th FOCS, pp. 1480–1498. IEEE Computer Society Press, October 2015
- [BR93] Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 1993, pp. 62–73. ACM Press, November 1993
- [BSS99] Blake, I., Seroussi, G., Smart, N.: *Elliptic Curves in Cryptography*, vol. 265. Cambridge University Press, Cambridge (1999)
- [CCH+19] Canetti, R., et al.: Fiat-Shamir: from practice to theory. In: Charikar, M., Cohen, E. (eds.) 51st ACM STOC, pp. 1082–1090. ACM Press, June 2019
- [CCR16] Canetti, R., Chen, Y., Reyzin, L.: On the correlation intractability of obfuscated pseudorandom functions. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A, Part I. LNCS, vol. 9562, pp. 389–415. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49096-9\\_17](https://doi.org/10.1007/978-3-662-49096-9_17)
- [CCRR18] Canetti, R., Chen, Y., Reyzin, L., Rothblum, R.D.: Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 91–122. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-78381-9\\_4](https://doi.org/10.1007/978-3-319-78381-9_4)
- [CDT09] Chen, X., Deng, X., Teng, S.-H.: Settling the complexity of computing two-player Nash equilibria. *J. ACM (JACM)* **56**(3), 1–57 (2009)

- [CGH98] Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version). In: 30th ACM STOC, pp. 209–218. ACM Press, May 1998
- [CHK+19a] Choudhuri, A.R., Hubáček, P., Kamath, C., Pietrzak, K., Rosen, A., Rothblum, G.N.: Finding a Nash equilibrium is no easier than breaking Fiat-Shamir. In: Charikar, M., Cohen, E. (eds.) 51st ACM STOC, pp. 1103–1114. ACM Press, June 2019
- [CHK+19b] Choudhuri, A.R., Hubáček, P., Kamath, C., Pietrzak, K., Rosen, A., Rothblum, G.N.: PPAD-hardness via iterated squaring modulo a composite. *Cryptology ePrint Archive*, Report 2019/667 (2019). <https://eprint.iacr.org/2019/667>
- [CJJ21] Choudhuri, A.R., Jain, A., Jin, Z.: Non-interactive batch arguments for NP from standard assumptions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 394–423. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84259-8\\_14](https://doi.org/10.1007/978-3-030-84259-8_14)
- [CJJ22] Choudhuri, A.R., Jain, A., Jin, Z.: SNARGs for P from LWE. In: 2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS), pp. 68–79. IEEE (2022)
- [CZ81] Cantor, D.G., Zassenhaus, H.: A new algorithm for factoring polynomials over finite fields. *Math. Comput.* **36**, 587–592 (1981)
- [DGP09] Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H.: The complexity of computing a Nash equilibrium. *SIAM J. Comput.* **39**(1), 195–259 (2009)
- [EFKP20] Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: Canteaut, A., Ishai, Y. (eds.) Annual International Conference on the Theory and Applications of Cryptographic Techniques, vol. 12107, pp. 125–154. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45727-3\\_5](https://doi.org/10.1007/978-3-030-45727-3_5)
- [FGK+10] Freeman, D.M., Goldreich, O., Kiltz, E., Rosen, A., Segev, G.: More constructions of lossy and correlation-secure trapdoor functions. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 279–295. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-13013-7\\_17](https://doi.org/10.1007/978-3-642-13013-7_17)
- [FS87] 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](https://doi.org/10.1007/3-540-47721-7_12)
- [GK03] Goldwasser, S., Kalai, Y.T.: On the (in)security of the Fiat-Shamir paradigm. In: 44th FOCS, pp. 102–115. IEEE Computer Society Press, October 2003
- [GKR08] Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 113–122. ACM Press, May 2008
- [Gol18] Goldreich, O.: On doubly-efficient interactive proof systems. *Found. Trends® Theor. Comput. Sci.* **13**(3), 158–246 (2018)
- [GPS16] Garg, S., Pandey, O., Srinivasan, A.: Revisiting the cryptographic hardness of finding a Nash equilibrium. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 579–604. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53008-5\\_20](https://doi.org/10.1007/978-3-662-53008-5_20)
- [GW11] Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC, pp. 99–108. ACM Press, June 2011

- [GZ21] González, A., Zacharakis, A.: Fully-succinct publicly verifiable delegation from constant-size assumptions. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13042, pp. 529–557. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-90459-3\\_18](https://doi.org/10.1007/978-3-030-90459-3_18)
- [HAB02] Hesse, W., Allender, E., Barrington, D.A.M.: Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.* **65**(4), 695–716 (2002)
- [HJKS22] Hulett, J., Jawale, R., Khurana, D., Srinivasan, A.: SNARGs for P from sub-exponential DDH and QR. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 520–549. Springer, Heidelberg (2022). [https://doi.org/10.1007/978-3-031-07085-3\\_18](https://doi.org/10.1007/978-3-031-07085-3_18)
- [HL18] Holmgren, J., Lombardi, A.: Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications). In: Thorup, M. (ed.) 59th FOCS, pp. 850–858. IEEE Computer Society Press, October 2018
- [HLR21] Holmgren, J., Lombardi, A., Rothblum, R.D.: Fiat-Shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge). In: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, pp. 750–760 (2021)
- [HV06] Healy, A., Viola, E.: Constant-depth circuits for arithmetic in finite fields of characteristic two. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 672–683. Springer, Heidelberg (2006). [https://doi.org/10.1007/11672142\\_55](https://doi.org/10.1007/11672142_55)
- [JJ21] Jain, A., Jin, Z.: Non-interactive zero knowledge from sub-exponential DDH. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 3–32. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77870-5\\_1](https://doi.org/10.1007/978-3-030-77870-5_1)
- [JKKZ21] Jawale, R., Kalai, Y.T., Khurana, D., Zhang, R.: SNARGs for bounded depth computations and ppad hardness from sub-exponential LWE. In: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, pp. 708–721 (2021)
- [JLS21] Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from well-founded assumptions. In: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, pp. 60–73 (2021)
- [KLVW23] Kalai, Y.T., Lombardi, A., Vaikuntanathan, V., Wichs, D.: Boosting batch arguments and ram delegation. In: STOC (2023). <https://eprint.iacr.org/2022/1320>
- [KPY18] Kalai, Y.T., Paneth, O., Yang, L.: On publicly verifiable delegation from standard assumptions. Cryptology ePrint Archive, Report 2018/776 (2018). <https://eprint.iacr.org/2018/776>
- [KPY19] Kalai, Y.T., Paneth, O., Yang, L.: How to delegate computations publicly. In: Charikar, M., Cohen, E. (eds.) 51st ACM STOC, pp. 1115–1124. ACM Press, June 2019
- [KPY20] Kalai, Y.T., Paneth, O., Yang, L.: Delegation with updatable unambiguous proofs and PPAD-hardness. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 652–673. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-56877-1\\_23](https://doi.org/10.1007/978-3-030-56877-1_23)
- [KRR17] Kalai, Y.T., Rothblum, G.N., Rothblum, R.D.: From obfuscation to the security of Fiat-Shamir for proofs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 224–251. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63715-0\\_8](https://doi.org/10.1007/978-3-319-63715-0_8)

- [KVZ21] Kalai, Y.T., Vaikuntanathan, V., Zhang, R.Y.: Somewhere statistical soundness, post-quantum security, and SNARGs. In: Nissim, K., Waters, B. (eds.) TCC 2021, Part I. LNCS, vol. 13042, pp. 330–368. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-90459-3\\_12](https://doi.org/10.1007/978-3-030-90459-3_12)
- [Lag70] Lagrange, J.-L.: Reflexions sur la resolution algebrique des equations, nouveaux memoires de l’acade. Royale des sciences et belles-lettres, avec l’histoire pour la meme annee **1**, 134–215 (1770)
- [LFKN90] Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. In: 31st FOCS, pp. 2–10. IEEE Computer Society Press, October 1990
- [LV20] Lombardi, A., Vaikuntanathan, V.: Fiat-Shamir for repeated squaring with applications to PPAD-hardness and VDFs. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 632–651. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-56877-1\\_22](https://doi.org/10.1007/978-3-030-56877-1_22)
- [Mei13] Meir, O.: IP = PSPACE using error-correcting codes. SIAM J. Comput. **42**(1), 380–403 (2013)
- [Mic94] Micali, S.: CS proofs (extended abstracts). In: 35th FOCS, pp. 436–453. IEEE Computer Society Press, November 1994
- [Pap94] Papadimitriou, C.H.: On the complexity of the parity argument and other inefficient proofs of existence. J. Comput. Syst. Sci. **48**(3), 498–532 (1994)
- [PS19] Peikert, C., Shiehian, S.: Noninteractive Zero Knowledge for NP from (Plain) Learning with Errors. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 89–114. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-26948-7\\_4](https://doi.org/10.1007/978-3-030-26948-7_4)
- [PW08] Peikert, C., Waters, B.: Lossy trapdoor functions and their applications. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 187–196. ACM Press, May 2008
- [Rab80] Rabin, M.O.: Probabilistic algorithms in finite fields. SIAM J. Comput. **9**(2), 273–280 (1980)
- [RRR16] Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC, pp. 49–62. ACM Press, June 2016
- [Tov84] Tovey, C.A.: A simplified np-complete satisfiability problem. Discrete Appl. Math. **8**(1), 85–89 (1984)
- [VV85] Valiant, L.G., Vazirani, V.V.: NP is as easy as detecting unique solutions. In: 17th ACM STOC, pp. 458–463. ACM Press, May 1985
- [WW22] Waters, B., Wu, D.J.: Batch arguments for np and more from standard bilinear group assumptions. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. CRYPTO 2022. LNCS, vol. 13508. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-15979-4\\_15](https://doi.org/10.1007/978-3-031-15979-4_15)





# HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates

Binyi Chen<sup>1(✉)</sup>, Benedikt Bünz<sup>1,2</sup>, Dan Boneh<sup>2</sup>, and Zhenfei Zhang<sup>1</sup>

<sup>1</sup> Espresso Systems, Menlo Park, USA  
binyi@espressosys.com

<sup>2</sup> Stanford University, Stanford, USA

**Abstract.** Plonk is a widely used succinct non-interactive proof system that uses univariate polynomial commitments. Plonk is quite flexible: it supports circuits with low-degree “custom” gates as well as circuits with lookup gates (a lookup gate ensures that its input is contained in a pre-defined table). For large circuits, the bottleneck in generating a Plonk proof is the need for computing a large FFT.

We present **HyperPlonk**, an adaptation of Plonk to the boolean hypercube, using multilinear polynomial commitments. **HyperPlonk** retains the flexibility of Plonk but provides several additional benefits. First, it avoids the need for an FFT during proof generation. Second, and more importantly, it supports custom gates of much higher degree than Plonk without harming the running time of the prover. Both of these can dramatically speed up the prover’s running time. Since **HyperPlonk** relies on multilinear polynomial commitments, we revisit two elegant constructions: one from **Orion** and one from **Virgo**. We show how to reduce the **Orion** opening proof size to less than 10 KB (an almost factor 1000 improvement) and show how to make the **Virgo** FRI-based opening proof simpler and shorter (This is an extended abstract. The full version is available on EPRINT[22]).

## 1 Introduction

Proof systems [4, 31] have a long and rich history in cryptography and complexity theory. In recent years, the efficiency of proof systems has dramatically improved and this has enabled a multitude of new real-world applications that were not previously possible. In this paper, we focus on succinct non-interactive arguments of knowledge, also called SNARKs [13]. Here, succinct refers to the fact that the proof is short and verification time is fast, as explained below. Recent years have seen tremendous progress in improving the efficiency of the prover [17, 27, 32, 43, 48–50, 55].

Let us briefly review what a (preprocessing) SNARK is. We give a precise definition in the full version. Fix a finite field  $\mathbb{F}$ , and consider the relation  $\mathcal{R}(\mathcal{C}, \mathbf{x}, \mathbf{w})$  that is true whenever  $\mathbf{x} \in \mathbb{F}^n$ ,  $\mathbf{w} \in \mathbb{F}^m$ , and  $\mathcal{C}(\mathbf{x}, \mathbf{w}) = 0$ , where  $\mathcal{C}$  is the description of an arithmetic circuit over  $\mathbb{F}$  that takes  $n + m$  inputs. A SNARK enables a prover  $\mathcal{P}$  to non-interactively and succinctly convince a verifier  $\mathcal{V}$  that  $\mathcal{P}$  knows

a witness  $w \in \mathbb{F}^m$  such that  $\mathcal{R}(\mathcal{C}, \mathbf{x}, w)$  holds, for some public circuit  $\mathcal{C}$  and  $\mathbf{x} \in \mathbb{F}^n$ .

In more detail, a SNARK is a tuple of four algorithms ( $\text{Setup}, \mathcal{I}, \mathcal{P}, \mathcal{V}$ ), where  $\text{Setup}(1^\lambda)$  is a randomized algorithm that outputs parameters  $\text{gp}$ , and  $\mathcal{I}(\text{gp}, \mathcal{C})$  is a deterministic algorithm that pre-processes the circuit  $\mathcal{C}$  and outputs prover parameters  $\text{pp}$  and verifier parameters  $\text{vp}$ . The prover  $\mathcal{P}(\text{pp}, \mathbf{x}, w)$  is a randomized algorithm that outputs a proof  $\pi$ , and the verifier  $\mathcal{V}(\text{vp}, \mathbf{x}, \pi)$  is a deterministic algorithm that outputs 0 or 1. The SNARK must be *complete*, *knowledge sound*, and *succinct*. Here *succinct* means that if  $\mathcal{C}$  contains  $s$  gates, and  $\mathbf{x} \in \mathbb{F}^n$ , then the size of the proof should be  $O_\lambda(\log s)$  and the verifier's running time should be  $\tilde{O}_\lambda(n + \log s)$ . A SNARK is often set in the random oracle model where all four algorithms can query the oracle. If the  $\text{Setup}$  algorithm is randomized, then we say that the SNARK requires a *trusted setup*; otherwise, the SNARK is said to be *transparent* because  $\text{Setup}$  only has access to public randomness via the random oracle. Optionally, we might want the SNARK to be zero-knowledge, in which case it is called a zkSNARK.

Modern SNARKs are constructed by compiling an information-theoretic object called an Interactive Oracle Proof (IOP) [11] to a SNARK using a suitable cryptographic commitment scheme. There are several examples of this paradigm. Some SNARKs use a univariate polynomial commitment scheme to compile a Polynomial-IOP to a SNARK. Examples include Marlin [24], and Plonk [27]. Other SNARKs use a multivariate linear (multilinear) commitment scheme to compile a multilinear-IOP to a SNARK. Examples include Hyrax [48], Libra [49], Spartan [43], Quarks [44], and Gemini [17]. Yet other SNARKs use a vector commitment scheme (such as a Merkle tree) to compile a vector-IOP to a SNARK. The STARK system [8] is the prime example in this category, but other examples include Aurora [10], Virgo [55], Brakedown [32], and Orion [51]. While STARKs are post-quantum secure, require no trusted setup, and have an efficient prover, they generate a relatively long proof (tens of kilobytes in practice). The paradigm of compiling an IOP to a SNARK using a suitable commitment scheme lets us build *universal* SNARKs where a single trusted setup can support many circuits. In earlier SNARKs, such as [14, 30, 34], every circuit required a new trusted setup.

*The Plonk System.* Among the IOP-based SNARKs that use a Polynomial-IOP, the Plonk system [27] has emerged as one of the most widely adopted in industry. This is because Plonk proofs are very short (about 400 bytes in practice) and fast to verify. Moreover, Plonk supports custom gates, as we will see in a minute. An extension of Plonk, called PlonKup [41], further extends Plonk to incorporate lookup gates using the Plookup IOP of [27].

One difficulty with Plonk, compared to some other schemes, is the prover's complexity. For a circuit  $\mathcal{C}$  with  $s$  arithmetic gates, the Plonk prover runs in time  $O_\lambda(s \log s)$ . The primary bottlenecks come from the fact that the prover must commit to and later open several degree  $O(s)$  polynomials. When using the KZG polynomial commitment scheme [36], the prover must (i) compute a multi-exponentiation of size  $O(s)$  in a pairing-friendly group where discrete log is hard, and (ii) compute several FFTs and inverse-FFT of dimension  $O(s)$ .

When using a FRI-based polynomial commitment scheme [7, 37, 55], the prover computes an  $O(cs)$ -sized FFT and  $O(cs)$  hashes, where  $1/c$  is the rate of a certain Reed-Solomon code. The performance further degrades for circuits that contain *high-degree* custom gates, as some FFTs and multi-exponentiations have size proportional to the degree of the custom gates.

In practice, when the circuit size  $s$  is bigger than  $2^{20}$ , the FFTs become a significant part of the running time. This is due to the quasi-linear running time of the FFT algorithm, while other parts of the prover scale linearly in  $s$ . The reliance on FFT is a direct result of Plonk’s use of *univariate* polynomials. We note that some proof systems eliminate the need for an FFT by moving away from Plonk altogether [17, 25, 32, 43, 51].

*Hyperplonk.* In this paper, we introduce HyperPlonk, an adaptation of the Plonk IOP and its extensions to operate over the boolean hypercube  $B_\mu := \{0, 1\}^\mu$ . We present HyperPlonk as a multilinear-IOP, which means that it can be compiled using a suitable multilinear commitment scheme to obtain a SNARK (or a zkSNARK) with an efficient prover.

HyperPlonk inherits the flexibility of Plonk to support circuits with custom gates, but presents several additional advantages. First, by moving to the boolean hypercube we eliminate the need for an FFT during proof generation. We do so by making use of the classic SumCheck protocol [39], and this reduces the prover’s running time from  $O_\lambda(s \log s)$  to  $O_\lambda(s)$ . The efficiency of SumCheck is the reason why many of the existing multilinear SNARKs [17, 43, 44, 48, 49] use the boolean hypercube. Here we show that Plonk can similarly benefit from the SumCheck protocol.

Second, and more importantly, we show that the hypercube lets us incorporate custom gates more efficiently into HyperPlonk. A custom gate is a function  $G : \mathbb{F}^\ell \rightarrow \mathbb{F}$ , for some  $\ell$ . An arithmetic circuit  $\mathcal{C}$  with a custom gate  $G$ , denoted  $\mathcal{C}[G]$ , is a circuit with addition and multiplication gates along with a custom gate  $G$  that can appear many times in the circuit. The circuit may contain multiple types of custom gates, but for now, we will restrict to one type to simplify the presentation. These custom gates can greatly reduce the circuit size needed to compute a function, leading to a faster prover. For example, if one needs to implement the S-box in a block cipher, it can be more efficient to implement it as a custom gate.

Custom gates are not free. Let  $G : \mathbb{F}^\ell \rightarrow \mathbb{F}$  be a custom gate that computes a multivariate polynomial of total degree  $d$ . Let  $\mathcal{C}[G]$  be a circuit with a total of  $s$  gates. In the Plonk IOP, the circuit  $\mathcal{C}[G]$  results in a prover that manipulates univariate polynomials of degree  $O(s \cdot d)$ . Consequently, when compiling Plonk using KZG [36], the prover needs to do a group multi-exponentiation of size  $O(sd)$  as well as FFTs of this dimension. This restricts custom gates in Plonk to gates of low degree.

We show that the prover’s work in HyperPlonk is much lower. Let  $G : \mathbb{F}^\ell \rightarrow \mathbb{F}$  be a custom gate that can be evaluated using  $k$  arithmetic operations. In HyperPlonk, the bulk of the prover’s work when processing  $\mathcal{C}[G]$  is only  $O(sk \log^2 k)$  field operations. Moreover, when using KZG multilinear commitments [40], the

total number of group exponentiations is only  $O(s + d \log s)$ , where  $d$  is the total degree of  $G$ . This is much lower than Plonk’s  $O(sd)$  group exponentiations. It lets us use custom gates of much higher degree in HyperPlonk.

Making Plonk and its Plonkup extension work over the hypercube raises interesting challenges, as discussed in Sect. 1.1. In particular, adapting the Plookup IOP [27], used to implement table lookups, requires changing the protocol to make it work over the hypercube (see Sect. 3.6). The resulting version of HyperPlonk that supports lookup gates is called HyperPlonk+ and is described in the full version.

*Batch Openings and Commit-and-Prove SNARKs.* The prover in HyperPlonk needs to open several multilinear polynomials at random points. We present a new sum-check-based batch-opening protocol (Sect. 3.7) that can batch many openings into one, significantly reducing the prover time, proof size, and verifier time. Our protocol takes  $O(k \cdot 2^\mu)$  field operations for the prover for batching  $k$  of  $\mu$ -variate polynomials, compared to  $O(k^2 \mu \cdot 2^\mu)$  for the previously best protocol [47]. Under certain conditions, we also obtain a more efficient batching scheme with complexity  $O(2^\mu)$ , which yields a very efficient commit-and-prove protocol.

*Improved Multilinear Commitments.* Since HyperPlonk relies on a multilinear commitment scheme, we revisit two approaches to constructing multilinear commitments and present significant improvements to both.

First, in Sect. 5 we use our commit-and-prove protocol to improve the Orion multilinear commitment scheme [51]. Orion is highly efficient: the prover time is strictly linear, taking only  $O(2^\mu)$  field operations and hashes for a multilinear polynomial in  $\mu$  variables (no group exponentiations are used). The proof size is  $O(\lambda \mu^2)$  hash and field elements, and the verifier time is proportional to the proof size. In Sect. 5 we describe Orion+, that has the same prover complexity, but has  $O(\mu)$  proof size and  $O(\mu)$  verifier time, with good constants. In particular, for security parameter  $\lambda = 128$  and  $\mu = 25$  the proof size with Orion+ is only about 7 KBs, compared with 5.5 MB with Orion, a nearly 1000x improvement. Using Orion+ in HyperPlonk gives a strictly linear time prover.

Second, in the full version, we show how to generically transform a univariate polynomial commitment scheme into a multilinear commitment scheme using the tensor-product *univariate* Polynomial-IOP from [17]. This yields a new construction for multilinear commitments from FRI [7] by applying the transformation to the univariate FRI-based commitment scheme from [37]. This approach leads to a more efficient FRI-based multilinear commitment scheme compared to the prior construction in [55], which uses recursive techniques. Using this commitment scheme in HyperPlonk gives a quantum-resistant quasilinear-time prover.

*Evaluation Results.* When optimized and instantiated with the pairing-based multilinear commitment scheme of [40], the proof size of Hyperplonk is  $\mu + 5$  group elements and  $4\mu + 29$  field elements<sup>1</sup>. Using BLS12-381 as the pairing

<sup>1</sup> The constants depend linearly on the degree of the custom gates. These numbers are for simple degree 2 arithmetic circuits.

group, we obtain 4.7KB proofs for  $\mu = 20$  and 5.5KB proofs for  $\mu = 25$ . For comparison, Kopis [44] and Gemini [17], which also have linear-time provers, report proofs of size 39KB and 18KB respectively for  $\mu = 20$ . In Table 1 we show that our prototype HyperPlonk implementation outperforms an optimized commercial-strength Plonk system for circuits with more than  $2^{14}$  gates. It also shows the effects of PLONK arithmetization compared to R1CS by comparing the prover runtime for several important applications. Hyperplonk outperforms Spartan [43] for these applications by a factor of over 60. We discuss the evaluation further in the full version.

**Table 1.** The prover runtime of Hyperplonk, Spartan [43], and Jellyfish Plonk, for popular applications. The first column (next to the column of the applications) shows the number of R1CS constraints for each application. The third column shows the corresponding number of constraints in HyperPlonk+. Note that the Zexe and the Rollup applications are using the BW6-761 curve.

Application	$\mathcal{R}_{\text{R1CS}}$	Spartan	$\mathcal{R}_{\text{PLONK+}}$	Jellyfish	HyperPlonk
3-to-1 Rescue Hash	288 [1]	422 ms	144 [45]	40 ms	88 ms
Zexe’s recursive circuit	$2^{22}$ [52]	6 min	$2^{17}$ [52]	13.1 s	5.1 s
Rollup of 50 private tx	$2^{25}$	39 min	$2^{20}$ [45]	110 s	38.2 s

## 1.1 Technical Overview

In this section we give a high level overview of how to make Plonk and its extensions work over the hypercube. We begin by describing Plonk in a modular way, breaking it down into a sequence of elementary components. In Sect. 3 we show how to instantiate each component over the hypercube.

Some components of Plonk rely on the simple linear ordering of the elements of a finite cyclic group induced by the powers of a generator. On the hypercube there is no natural simple ordering, and this causes a problem in the Plookup protocol [27] that is used to implement a lookup gate. To address this we modify the Plookup argument in Sect. 3.6 to make it work over the hypercube. We give an overview of our approach below.

*A Review of Plonk.* Let us briefly review the Plonk SNARK. Let  $\mathcal{C}[G] : \mathbb{F}^{n+m} \rightarrow \mathbb{F}$  be a circuit with a total of  $s$  gates, where each gate has fan-in two and can be one of addition, multiplication, or a custom gate  $G : \mathbb{F}^2 \rightarrow \mathbb{F}$ . Let  $\mathbf{x} \in \mathbb{F}^n$  be a public input to the circuit. Plonk represents the resulting computation as a sequence of  $n + s + 1$  triples<sup>2</sup>:

$$\hat{M} := \left\{ (L_i, R_i, O_i) \in \mathbb{F}^3 \right\}_{i=0, \dots, n+s}. \quad (1)$$

<sup>2</sup> A more general Plonkish arithmetization [54] supports wider tuples, but triples are sufficient here.

This  $\hat{M}$  is a matrix with three columns and  $n + s + 1$  rows. The first  $n$  rows encode the  $n$  public input; the next  $s$  rows represent the left and right inputs and the output for each gate; and the final row enforces that the final output of the circuit is zero. We will see how in a minute.

In basic (univariate) Plonk, the prover encodes the cells of  $\hat{M}$  using a cyclic subgroup  $\Omega \subseteq \mathbb{F}$  of order  $3(n + s + 1)$ . Specifically, let  $\omega \in \Omega$  be a generator. Then the prover interpolates and commits to a polynomial  $M \in \mathbb{F}[X]$  such that

$$M(\omega^{3i}) = L_i, \quad M(\omega^{3i+1}) = R_i, \quad M(\omega^{3i+2}) = O_i \quad \text{for } i = 0, \dots, n + s.$$

Now the prover needs to convince the verifier that the committed  $M$  encodes a valid computation of the circuit  $\mathcal{C}$ . This is the bulk of Plonk system.

*Hyperplonk.* In HyperPlonk we instead use the boolean hypercube to encode  $\hat{M}$ . From now on, suppose that  $n + s + 1$  is a power of two, so that  $n + s + 1 = 2^\mu$ . The prover interpolates and commits to a multilinear polynomial  $M \in \mathbb{F}[X^{\mu+2}] = \mathbb{F}[X_1, \dots, X_{\mu+2}]$  such that

$$M(0, 0, \langle i \rangle) = L_i, \quad M(0, 1, \langle i \rangle) = R_i, \quad M(1, 0, \langle i \rangle) = O_i, \quad \text{for } i = 0, \dots, n + s. \quad (2)$$

Here  $\langle i \rangle$  is the  $\mu$ -bit binary representation of  $i$ . Note that a multilinear polynomial on  $\mu + 2$  variables is defined by a vector of  $2^{\mu+2} = 4 \times 2^\mu$  coefficients. Hence, it is always possible to find a multilinear polynomial that satisfies the  $3 \times 2^\mu$  constraints in Eq. (2). Next, the prover needs to convince the verifier that the committed  $M$  encodes a valid computation of the circuit  $\mathcal{C}$ . To do so, we need to adapt Plonk to work over the hypercube.

Let us start with the pre-processing algorithm  $\mathcal{I}(\text{gp}, \mathcal{C})$  that outputs prover and verifier parameters  $\text{pp}$  and  $\text{vp}$ . The verifier parameters  $\text{vp}$  encode the circuit  $\mathcal{C}[G]$  as a commitment to four multilinear polynomials  $(S_1, S_2, S_3, \sigma)$ , where  $S_1, S_2, S_3 \in \mathbb{F}[X^\mu]$  and  $\sigma \in \mathbb{F}[X^{\mu+2}]$ . The first three are called *selector polynomials* and  $\sigma$  is called the *wiring polynomial*. We will see how they are defined in a minute. There is one more auxiliary multilinear polynomial  $I \in \mathbb{F}[X^\mu]$  that encodes the input  $\mathbf{x} \in \mathbb{F}^n$ . This polynomial is defined as  $I(\langle i \rangle) = \mathbf{x}_i$  for  $i = 0, \dots, n - 1$ , and is zero on the rest of the boolean cube  $B_\mu$ . The verifier, on its own, computes a commitment to the polynomial  $I$  to ensure that the correct input  $\mathbf{x} \in \mathbb{F}^n$  is being used in the proof. Computing a commitment to  $I$  can be done in time  $O_\lambda(n)$ , which is within the verifier’s time budget.

With this setup, the Plonk prover  $\mathcal{P}$  convinces the verifier that the committed  $M$  satisfies two polynomial identities:

*The Gate Identity:* Let  $S_1, S_2, S_3 : \mathbb{F}^\mu \rightarrow \{0, 1\}$  be the three selector polynomials that the pre-processing algorithm  $\mathcal{I}(\text{gp}, \mathcal{C})$  committed to in  $\text{vp}$ . To prove that all gates were evaluated correctly, the prover convinces the verifier that the following

identity holds for all  $\mathbf{x} \in B_\mu := \{0, 1\}^\mu$ :

$$\begin{aligned}
 0 = & S_1(\mathbf{x}) \cdot \left( \underbrace{M(0, 0, \mathbf{x})}_{L_{[\mathbf{x}]}} + \underbrace{M(0, 1, \mathbf{x})}_{R_{[\mathbf{x}]}} \right) + S_2(\mathbf{x}) \cdot \underbrace{M(0, 0, \mathbf{x})}_{L_{[\mathbf{x}]}} \cdot \underbrace{M(0, 1, \mathbf{x})}_{R_{[\mathbf{x}]}} \\
 & + S_3(\mathbf{x}) \cdot G \left( \underbrace{M(0, 0, \mathbf{x})}_{L_{[\mathbf{x}]}} , \underbrace{M(0, 1, \mathbf{x})}_{R_{[\mathbf{x}]}} \right) - \underbrace{M(1, 0, \mathbf{x})}_{O_{[\mathbf{x}]}} + I(\mathbf{x})
 \end{aligned} \tag{3}$$

where  $[\mathbf{x}] = \sum_{i=0}^{\mu-1} \mathbf{x}_i 2^i$  is the integer whose binary representation is  $\mathbf{x} \in B_\mu$ . For each  $i = 0, \dots, n + s$ , the selector polynomials  $S_1, S_2, S_3$  are defined to do the “right” thing:

- addition gate:  $S_1(\langle i \rangle) = 1, \quad S_2(\langle i \rangle) = S_3(\langle i \rangle) = 0 \quad (O_i = L_i + R_i)$
- multiplication gate:  $S_1(\langle i \rangle) = S_3(\langle i \rangle) = 0, \quad S_2(\langle i \rangle) = 1 \quad (O_i = L_i \cdot R_i)$
- for a  $G$  gate:  $S_1(\langle i \rangle) = S_2(\langle i \rangle) = 0, \quad S_3(\langle i \rangle) = 1 \quad (O_i = G(L_i, R_i))$
- if  $i < n$  or  $i = n + s$ :  $S_1(\langle i \rangle) = S_2(\langle i \rangle) = S_3(\langle i \rangle) = 0 \quad (O_i = I(\langle i \rangle))$ .

The last bullet ensures that  $O_i$  is equal to the  $i$ -th input for  $i = 0, \dots, n - 1$ , and that the final output of the circuit,  $O_{n+s}$ , is equal to zero.

*The Wiring Identity:* Every wire in the circuit  $\mathcal{C}$  induces an equality constraint on two cells in the matrix  $M$ . In HyperPlonk, the wiring constraints are captured by a permutation  $\hat{\sigma} : B_{\mu+2} \rightarrow B_{\mu+2}$ . The prover needs to convince the verifier that

$$M(\mathbf{x}) = M(\hat{\sigma}(\mathbf{x})) \quad \text{for all } \mathbf{x} \in B_{\mu+2} := \{0, 1\}^{\mu+2}. \tag{4}$$

To do so, the pre-processing algorithm  $\mathcal{I}(\text{gp}, \mathcal{C})$  commits to a multilinear polynomial  $\sigma : \mathbb{F}^{\mu+2} \rightarrow \mathbb{F}$  that satisfies  $\sigma(\mathbf{x}) = [\hat{\sigma}(\mathbf{x})]$  for all  $\mathbf{x} \in B_{\mu+2}$  (recall that  $[\hat{\sigma}(\mathbf{x})]$  is the integer whose binary representation is  $\hat{\sigma}(\mathbf{x}) \in B_{\mu+2}$ ). The prover then convinces the verifier that the following two sets are equal (both sets are subsets of  $\mathbb{F}^2$ ):

$$\left\{ ([\mathbf{x}], M(\mathbf{x})) \right\}_{\mathbf{x} \in B_{\mu+2}} = \left\{ ([\hat{\sigma}(\mathbf{x})], M(\mathbf{x})) \right\}_{\mathbf{x} \in B_{\mu+2}}. \tag{5}$$

This equality of sets implies that Eq. (4) holds.

*Proving the Gate Identity.* The prover convinces the verifier that the Gate identity holds by proving that the polynomial defined by the right hand side of Eq. (3) is zero for all  $\mathbf{x} \in B_\mu$ . This is done using a ZeroCheck IOP, defined in Sect. 3.2. If the custom gate  $G$  has total degree  $d$  and there are  $s$  gates in the circuit, then the total number of terms of the polynomial in Eq. (3) is  $(d + 1)(s + n + 1)$  which is about  $(d \cdot s)$ . If this were a univariate polynomial, as in Plonk, then a ZeroCheck would require a multi-exponentiation of dimension  $(d \cdot s)$  and an FFT of the same dimension. When the polynomial is defined over the hypercube, the ZeroCheck is implemented using the SumCheck protocol in Sect. 3.1, which requires no FFTs. In that section we describe two optimizations to the SumCheck protocol for the settings where the multivariate polynomial has a high degree  $d$  in every variable:

- First, in every round of SumCheck the prover sends a polynomial commitment to a univariate polynomial of degree  $d$ , instead of sending the polynomial in the clear as in regular SumCheck. This greatly reduces the proof size.
- Second, in standard SumCheck, the prover opens the univariate polynomial commitment at three points: at 0, 1, and at a random  $r \in \mathbb{F}$ . We optimize this step by showing that opening the commitment at a *single* point is sufficient. This further shortens the final proof.

The key point is that the resulting ZeroCheck requires the prover to do only about  $s + d \cdot \mu$  group exponentiations, which is much smaller than  $d \cdot s$  in Plonk. The additional arithmetic work that the prover needs to do depends on the number of multiplication gates in the circuit implementing the custom gate  $G$ , not on the total degree of  $G$ , as in Plonk. As such, we can support much larger custom gates than Plonk.

In summary, proof generation time is reduced for two reasons: (i) the elimination of the FFTs, and (ii) the better handling of high-degree custom gates.

*Proving the Wiring Identity.* The prover convinces the verifier that the Wiring identity holds by proving the set equality in Eq. (5). We describe a set equality protocol over the hypercube in Sect. 3.4. Briefly, we use a technique from Bayer and Groth [6], that is also used in Plonk, to reduce this problem to a certain ProductCheck over the hypercube (Sect. 3.3). We then use an idea from Quarks [44] to reduce the hypercube ProductCheck to a ZeroCheck, which then reduces to a SumCheck. Again, no FFTs are needed.

*Table Lookups.* An important extension to Plonk supports circuits with table lookup gates. The table is represented as a fixed vector  $\mathbf{t} \in \mathbb{F}^{2^\mu - 1}$ . A table lookup gate ensures that a specific cell in the matrix  $\hat{M}$  is contained in  $\mathbf{t}$ . For example, one can set  $\mathbf{t}$  to be the field elements in  $\{0, 1, \dots, B\}$  for some  $B$  (padding the vector by 0 as needed). Now, checking that a cell in  $\hat{M}$  is contained in  $\mathbf{t}$  is a simple way to implement a range check.

Let  $f, t : B_\mu \rightarrow \mathbb{F}$  be two multilinear polynomials. Here the polynomial  $t$  encodes the table  $\mathbf{t}$ , where the table values are  $t(B_\mu)$ . The polynomial  $f$  encodes the cells of  $\hat{M}$  that need to be checked. An important step in supporting lookup gates in Plonk is a way for the prover to convince the verifier that  $f(B_\mu) \subseteq t(B_\mu)$ , when the verifier has commitments to  $f$  and  $t$ . The Plookup proof system by Gabizon and Williamson [27] is a way for the prover to do just that. More recently preprocessed alternatives to lookup have been developed [42, 53]. These perform better if the table is known, e.g. a range of values but are in general orthogonal to Plookup.

The problem is that Plookup is designed to work when the polynomials are defined over a cyclic subgroup  $\mathbb{G} \subseteq \mathbb{F}^*$  of order  $q$  with generator  $\omega \in \mathbb{G}$ . In particular, Plookup requires a function  $\text{next} : \mathbb{F} \rightarrow \mathbb{F}$  that induces an ordering of  $\mathbb{G}$ . This function must satisfy two properties: (i) the sequence

$$\omega, \text{ next}(\omega), \text{ next}(\text{next}(\omega)), \dots, \text{ next}^{(q-1)}(\omega) \tag{6}$$



should traverse all of  $\mathbb{G}$ , and (ii) the function  $\text{next}$  should be a *linear* function. This is quite easy in a cyclic group: simply define  $\text{next}(x) := \omega x$ .

To adapt **Plookup** to the hypercube we need a *linear* function  $\text{next} : \mathbb{F}^\mu \rightarrow \mathbb{F}^\mu$  that traverses all of  $B_\mu$  as in Eq. (6), starting with some element  $\mathbf{x}_0 \in B_\mu$ . However, such an  $\mathbb{F}$ -linear function does not exist. Nevertheless, we construct in Sect. 3.6 a quadratic function from  $\mathbb{F}^\mu$  to  $\mathbb{F}^\mu$  that traverses  $B_\mu$ . The function simulates  $B_\mu$  using a binary extension and has a beautiful connection to similar techniques used in early PCP work [12]. We then show how to linearize the function by modifying some of the building blocks that **Plookup** uses. This gives an efficient **Plookup** protocol over the hypercube. In the full version we use this hypercube **Plookup** protocol to support lookup gates in **HyperPlonk**. The resulting protocol is called **HyperPlonk+**.

## 2 Preliminaries

*Notation:* We use  $\lambda$  to denote the security parameter. For  $n \in \mathbb{N}$  let  $[n]$  be the set  $\{1, 2, \dots, n\}$ ; for  $a, b \in \mathbb{N}$  let  $[a, b]$  denote the set  $\{a, a + 1, \dots, b - 1\}$ . A function  $f(n)$  is  $\text{poly}(\lambda)(n)$  if there exists a  $c \in \mathbb{N}$  such that  $f(n) = O(n^c)$ . If for all  $c \in \mathbb{N}$ ,  $f(n)$  is  $o(n^{-c})$ , then  $f(n)$  is in  $\text{negl}(\lambda)(n)$  and is said to be **negligible**. A probability that is  $1 - \text{negl}(\lambda)(n)$  is **overwhelming**. We use  $\mathbb{F}$  to denote a field of prime order  $p$  such that  $\log(p) = \Omega(\lambda)$ .

A *multiset* is an extension of the concept of a set where every element has a positive multiplicity. Two finite multisets are equal if they contain the same elements with the same multiplicities.

A **relation** is a set of pairs  $(\mathbf{x}, \mathbf{w})$ . An **indexed relation** is a set of triples  $(i, \mathbf{x}; \mathbf{w})$ . The index  $i$  is fixed at setup time.

In defining the syntax of the various protocols, we use the following convention concerning public values (known to both the prover and the verifier) and secret ones (known only to the prover). In any list of arguments or returned tuple  $(a, b, c; d, e)$ , those variables listed before the semicolon are public, and those listed after it are secret. When there is no secret information, the semicolon is omitted.

### 2.1 Proofs and Arguments of Knowledge

We refer to the full version for more detailed definitions of proofs, arguments, and polynomial interactive oracle proofs. We briefly overview the primitives used and constructed in this paper.

Interactive proofs and arguments of knowledge consist of a non-interactive *preprocessing* phase run by an indexer and an interactive *online* phase between a prover and a verifier. They satisfy the notions of *completeness* and *knowledge soundness*, as well as optionally *zero-knowledge*. The protocols described in this paper are *public coin*, meaning that the verifier only sends random messages.

*PolyIOPs.* SNARKs can be constructed from information-theoretic proof systems that give the verifier oracle access to prover messages. The information-theoretic proof is then compiled using a cryptographic tool, such as a polynomial commitment. We now define a specific type of information-theoretic proof system called polynomial interactive oracle proofs (PIOPs). In a PIOP, the prover sends oracles to multi-variate polynomials as messages, and the verifier can query these polynomials at arbitrary points. The statement and the index can also consist of oracles to polynomials which the verifier can query. See the full version for formal definitions of PIOPs.

### 2.2 Multilinear Polynomial Commitments

Multilinear polynomial commitments are commitments where the message space is a multi-linear polynomial. It has the additional property that there exists an efficient argument of knowledge for convincing a verifier that the committed polynomial evaluates to a specific value at a given point.

Multi-linear polynomial commitments can be instantiated from random oracles using the FRI protocol [55], bilinear groups [40], groups of unknown order [3, 19] and discrete logarithm groups[18, 48]. We give a table of polynomial commitments with their different properties in Table 2:

**Table 2.** Multi-linear polynomial commitment schemes for  $\mu$ -variate linear polynomials and  $n = 2^\mu$ . The prover time measures the complexity of committing to a polynomial and evaluating it once. The commitment size is constant for all protocols. Unless constants are mentioned, the metrics are assumed to be asymptotic. In the 4th row,  $\rho$  denotes the rate of Reed-Solomon codes. In the 5th and 6th rows,  $k$  denotes the number of rows of the matrix that represents the polynomial coefficients. The 6th column measures the concrete proof size for  $n = 2^{25}$ , i.e.  $\mu = 25$  and 128-bit security. Legend: BL = Bilinear Group, DL = Discrete Logarithm, RO = Random Oracle,  $H$  = Hashes,  $P$  = pairings,  $\mathbb{G}$  = group scalar multiplications, rec. = Recursive circuit size, univ. = universal setup, trans. = transparent setup, Add. = Additive

Scheme		Prover time: Commit+ Eval	Verifier time	Proof size	$n = 2^{25}$	Setup	Add
KZG-based [40]	BL	$n \mathbb{G}_1$	$\log(n) P$	$\log(n) \mathbb{G}_1$	0.8 KB	Univ	Yes
Dory [38]	BL	$n\mathbb{G}_1 + \sqrt{n}P$	$\log(n) \mathbb{G}_T$	$6 \log(n) \mathbb{G}_T$	30 KB	Trans	Yes
Bulletproofs [18]	DL	$n \mathbb{G}$	$n \mathbb{G}$	$2 \log(n) \mathbb{G}$	1.6 KB	Trans	Yes
FRI-based [22]	RO	$n \log(n) / \rho^{\mathbb{F}} + n / \rho H$	$\log^2(n) \frac{\lambda}{-\log \rho} H$	$\log^2(n) \frac{\lambda}{-\log \rho} H$	250 KB	Trans	No
Orion	RO	$nH + \frac{n}{k} + k \text{ rec}$	$\lambda \log^2 nH$	$\lambda \log^2 n H$	5.5 MB	Trans	No
Orion + (§5)	BL	$n/k\mathbb{G}_1 + nH + (k\lambda H + \frac{n}{k} \mathbb{F}) \text{ rec.}$	$\log(n)P$	$4 \log n \mathbb{G}_1$	7 KB	Univ	No

*Virtual Oracles and Commitments.* Given multiple polynomial oracles, we can construct virtual oracles to the functions of these polynomials. An oracle to  $g([[f_1]], \dots, [[f_k]])$  for some function  $g$  is simply the list of oracles  $\{[[f_1]], \dots, [[f_k]]\}$  as well as a description of  $g$ . In order to evaluate  $g([[f_1]], \dots, [[f_k]])$  at some point  $\mathbf{x}$  we compute  $y_i = f_i(\mathbf{x}) \forall i \in [k]$  and output

$g(y_1, \dots, y_k)$ . Equivalently given commitments to polynomials, we can construct a virtual commitment to a function of these polynomials similarly. If  $g$  is an additive function and the polynomial commitment is additively homomorphic, then we can use the homomorphism to evaluate.

### 2.3 PIOP Compilation

PIOP compilation transforms the interactive oracle proof into an interactive argument of knowledge (without oracles)  $\Pi$ . The compilation replaces the oracles with polynomial commitments. Every query by the verifier is replaced with an invocation of the Eval protocol at the query point  $\mathbf{z}$ . The compiled verifier accepts if the PIOP verifier accepts and if the output of all Eval invocations is 1. If  $\Pi$  is public-coin, it can further be compiled into a non-interactive argument of knowledge (or NARK) using the Fiat-Shamir transform.

**Theorem 2.1 (PIOP Compilation [19, 24]).** *If the polynomial commitment scheme  $\Gamma$  has witness-extended emulation, and if the  $t$ -round Polynomial IOP for  $\mathcal{R}$  has negligible knowledge error, then  $\Pi$ , the output of the PIOP compilation, is a secure (non-oracle) argument of knowledge for  $\mathcal{R}$ . The compilation also preserves zero knowledge. If  $\Gamma$  is hiding and Eval is honest-verifier zero-knowledge, then  $\Pi$  is honest-verifier zero-knowledge. The efficiency of the resulting argument of knowledge  $\Pi$  depends on the efficiency of both the PIOP and  $\Gamma$ :*

- Prover time The prover time is equal to the prover time of the PIOP plus the oracle length times the commitment time plus the query complexity times the prover time of  $\Gamma$ .
- Verifier time The verifier time is equal to the verifier time of the PIOP plus the verifier time for  $\Gamma$  times the query complexity of the PIOP.
- Proof size The proof size is equal to the message complexity of the PIOP times the commitment size plus the query complexity times the proof size of  $\Gamma$ . We say the proof is succinct if the proof size is  $O(\log^c(|\mathbf{w}|))$ .

*Batching.* The prover time, verifier time, and proof size can be significantly reduced using batch openings of the polynomial commitments. For example, the proof size only depends on the number of oracles plus a single batch opening.

## 3 A Toolbox for Multivariate Polynomials

We begin by reviewing several important PolyIOPs that will serve as building blocks for HyperPlonk. Some are well-known, and some are new.

*Notation.* From here on, we let  $B_\mu := \{0, 1\}^\mu \subseteq \mathbb{F}^\mu$  be the boolean hypercube. We use  $\mathcal{F}_\mu^{(\leq d)}$  to denote the set of multivariate polynomials in  $\mathbb{F}[X_1, \dots, X_\mu]$  where the degree in each variable is at most  $d$ ; moreover, we require that each polynomial in  $\mathcal{F}_\mu^{(\leq d)}$  can be expressed as a virtual oracle to  $c = O(1)$  multilinear polynomials. that is, with the form  $f(\mathbf{X}) := g(h_1(\mathbf{X}), \dots, h_c(\mathbf{X}))$  where  $h_i \in$

$\mathcal{F}_\mu^{(\leq 1)}$  ( $1 \leq i \leq c$ ) is multilinear and  $g$  is a  $c$ -variate polynomial of total degree at most  $d$ . Looking ahead, we restrict ourselves to this kind of polynomials so that we can have sumchecks for the polynomials with linear-time provers.

For polynomials  $f, g \in \mathcal{F}_\mu^{(\leq d)}$ , we denote  $\text{merge}(f, g) \in \mathcal{F}_{\mu+1}^{(\leq d)}$  as

$$\text{merge}(f, g) := h(\mathbf{X}_0, \dots, \mathbf{X}_\mu) := (1 - \mathbf{X}_0) \cdot f(\mathbf{X}_1, \dots, \mathbf{X}_\mu) + \mathbf{X}_0 \cdot g(\mathbf{X}_1, \dots, \mathbf{X}_\mu) \quad (7)$$

so that  $h(0, \mathbf{X}) = f(\mathbf{X})$  and  $h(1, \mathbf{X}) = g(\mathbf{X})$ . In the following definitions, we omit the public parameters  $\mathbf{gp} := (\mathbb{F}, \mu, d)$  when the context is clear. We use  $\delta_{\text{check}}^{d, \mu}$  to denote the soundness error of the PolyIOP for relation  $\mathcal{R}_{\text{check}}$  with public parameter  $(\mathbb{F}, d, \mu)$ , where  $\text{check} \in \{\text{sum, zero, prod, mset, perm, lkup}\}$ .

We defer all proofs to the full version[22] (Table 3).

**Table 3.** The complexity of PIOPs.  $d$  and  $\mu$  denote the degree and the number of variables of the multivariate polynomials;  $k$  in MsetCheck is the length of each element in the multisets;  $k$  in BatchEval is the number of evaluations.

Scheme	$\mathcal{P}$ time	$\mathcal{V}$ time	Num of queries	Num of rounds	Proof oracle size	Witness size
SumCheck	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu + 1$	$\mu$	$d\mu$	$O(2^\mu)$
ZeroCheck	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu + 1$	$\mu$	$d\mu$	$O(2^\mu)$
ProdCheck	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu + 2$	$\mu + 1$	$O(2^\mu)$	$O(2^\mu)$
MsetEqChk	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu + 2$	$\mu + 1$	$O(2^\mu)$	$O(k2^\mu)$
PermCheck	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu + 2$	$\mu + 1$	$O(2^\mu)$	$O(2^\mu)$
Plookup	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu + 3$	$\mu + 2$	$O(2^\mu)$	$O(2^\mu)$
BatchEval	$O(2^\mu k)$	$O(k\mu)$	1	$\mu + \log k$	$O(\mu + \log k)$	$O(k2^\mu)$

### 3.1 SumCheck PIOP for High Degree Polynomials

In this section, we describe a PIOP for the sumcheck relation using the classic sumcheck protocol [39]. However, we modify the protocol and adapt it to our setting of high-degree polynomials.

**Definition 3.1 (SumCheck relation).** *The relation  $\mathcal{R}_{SUM}$  is the set of all tuples  $(\mathbf{x}; \mathbf{w}) = ((v, [[f]]); f)$  where  $f \in \mathcal{F}_\mu^{(\leq d)}$  and  $\sum_{\mathbf{b} \in B_\mu} f(\mathbf{b}) = v$ .*

*Construction.* The classic SumCheck protocol [39] is a PolyIOP for the relation  $\mathcal{R}_{SUM}$ . When applying the protocol to a polynomial  $f \in \mathcal{F}_\mu^{(\leq d)}$ , the protocol runs in  $\mu$  rounds where in every round, the prover sends a univariate polynomial of degree at most  $d$  to the verifier. The verifier then sends a random challenge point for the univariate polynomial. At the end of the protocol, the verifier checks the consistency between the univariate polynomials and the multi-variate polynomial using a single query to  $f$ .

Given a tuple  $(\mathbf{x}; \mathbf{w}) = (v, [[f]]); f)$  for  $\mu$ -variate degree  $d$  polynomial  $f$  such that  $\sum_{\mathbf{b} \in B_\mu} f(\mathbf{b}) = v$ :

- For  $i = \mu, \mu - 1, \dots, 1$ :

- The prover computes  $r_i(X) := \sum_{\mathbf{b} \in B_{i-1}} f(\mathbf{b}, X, \alpha_{i+1}, \dots, \alpha_\mu)$  and sends the oracle  $[[r_i]]$  to the verifier.  $r_i$  is univariate and of degree at most  $d$ .
  - The verifier checks that  $v = r_i(0) + r_i(1)$ , samples  $\alpha_i \leftarrow \mathbb{F}$ , sends  $\alpha_i$  to the prover, and sets  $v \leftarrow r_i(\alpha_i)$ .
- Finally, the verifier accepts if  $f(\alpha_1, \dots, \alpha_\mu) = v$ .

**Theorem 3.2.** *The PIOP for  $\mathcal{R}_{SUM}$  is perfectly complete and has knowledge error  $\delta_{sum}^{d,\mu} := d\mu/|\mathbb{F}|$ .*

We refer to [47] for the proof of the theorem.

*Sending  $r$  as an Oracle.* Unlike in the classic sumcheck protocol, we send an oracle to  $r_i$ , in each round, instead of the actual polynomial. This does not change the soundness analysis, as the soundness is still proportional to the degree of the univariate polynomials sent in each round. However, it reduces the communication and verifier complexity, especially if the degree of  $r$  is large, as in our application of Hyperplonk with custom gates.

Moreover, the verifier has to evaluate  $r_i$  at three points: 0, 1, and  $\alpha_i$ . As a useful optimization, the prover can instead send an oracle for the degree  $d - 2$  polynomial

$$r'_i(X) := \frac{r_i(X) - (1 - X) \cdot r_i(0) - X \cdot r_i(1)}{X \cdot (1 - X)},$$

along with  $r_i(0)$ . The verifier then computes  $r_i(1) \leftarrow v - r_i(0)$  and

$$r_i(\alpha_i) \leftarrow r'_i(\alpha_i) \cdot (1 - \alpha_i) \cdot \alpha_i + (1 - \alpha_i) \cdot r_i(0) + \alpha_i \cdot r_i(1).$$

This requires only one query to the oracle of  $r'_i$  at  $\alpha_i$  and one field element per round.

*Computing Sumcheck for High-Degree Polynomials.* Consider a multi-variate polynomial  $f(\mathbf{X}) := h(g_1(\mathbf{X}), \dots, g_c(\mathbf{X}))$  such that  $h$  is degree  $d$  and can be evaluated through an arithmetic circuit with  $O(d)$  gates. In the sumcheck protocol, the prover has to compute a univariate polynomial  $r_i(X)$  in each round using the previous verifier messages  $\alpha_1, \dots, \alpha_{i-1}$ . We adapt the algorithm by [46, 49] that showed how the sumcheck prover can be run in time linear in  $2^\mu$  using dynamic programming. The algorithm takes as input a description of  $f$  as well as the sumcheck round challenges  $\alpha_1, \dots, \alpha_\mu$ . It outputs the round polynomials  $r_1, \dots, r_\mu$ . The sumcheck prover runs the algorithm in parallel to the sumcheck protocol, taking each computed  $r_i$  as that rounds message:

---

**Algorithm 1.** Computing  $r_1, \dots, r_\mu$  [46, 49]

---

```

1: procedure SUMCHECK PROVER( $h, g_1(\mathbf{X}), \dots, g_c(\mathbf{X})$ )
2:   For each  $g_j$  build table  $A_j : \{0, 1\}^\mu \rightarrow \mathbb{F}$  of all evaluations over  $B_\mu$ 
3:   for  $i \leftarrow \mu \dots 1$  do
4:     For each  $\mathbf{b} \in B_{i-1}$  and each  $j \in [c]$ , define  $r^{(j,\mathbf{b})}(X) := (1 - X)A_j[\mathbf{b}, 0] + X A_j[\mathbf{b}, 1]$ .
5:     Compute  $r^{(b)}(X) \leftarrow h(r^{(1,\mathbf{b})}(X), \dots, r^{(c,\mathbf{b})}(X))$  for all  $\mathbf{b} \in B_{i-1}$  using Algorithm 2 .
6:      $r_i(X) \leftarrow \sum_{\mathbf{b} \in B_{i-1}} r_b(X)$ .
7:     Send  $r_i(X)$  to  $\mathcal{V}$ .
8:     Receive  $\alpha_i$  from  $\mathcal{V}$ .
9:     Set  $A_j[\mathbf{b}] \leftarrow r^{(j,\mathbf{b})}(\alpha_i)$  for each  $\mathbf{b} \in B_{i-1}$ .
10:  end for
11: end procedure

```

---

In [46, 49],  $r^{(b)}(X) := h(r^{(1,\mathbf{b})}(X), \dots, r^{(c,\mathbf{b})}(X))$  is computed by evaluating  $h$  on  $d$  distinct values for  $X$ , e.g.  $X \in \{0, \dots, d\}$  and interpolating the output. This works as  $h$  is a degree  $d$  polynomial and each  $r^{j,\mathbf{b}}$  is linear. Evaluating  $r^{j,\mathbf{b}}$  on  $d$  points can be done in  $d$  steps. So the total time to evaluate all  $r^{j,\mathbf{b}}$  for  $j \in [c]$  is  $c \cdot d$ . Furthermore, the circuit has  $O(d)$  gates, and evaluating it on  $d$  inputs, takes time  $O(d^2)$ . Assuming that  $c \approx d$  the total time to compute  $r^{(b)}$  with this algorithm is  $O(d^2)$  and the time to run Algorithm 1 is  $O(2^\mu d^2)$ .

We show how this can be reduced to  $O(2^\mu \cdot d \log^2 d)$  for certain low depth circuits, such as  $h := \prod_c r_c(\mathbf{X})$ . The core idea is that evaluating the circuit for  $h$  *symbolically*, instead of at  $d$  individual points, is faster if fast polynomial multiplication algorithms are used.

We will present the algorithm for computing  $h(X) := \prod_{j=1}^d r_j(X)$ , then we will discuss how to extend this for more general  $h$ . Assume w.l.o.g. that  $d$  is a power of 2.

---

**Algorithm 2.** Evaluating  $h := \prod_{j=1}^d r_j$

---

**Require:**  $r_1, \dots, r_d$  are linear functions

```

1: procedure  $h(r_1(X), \dots, r_d(X))$ 
2:    $t_{1,j} \leftarrow r_j$  for all  $j \in [d]$ .
3:   for  $i \leftarrow 1 \dots \log d$  do
4:     for  $j \in [d/2^i]$  do
5:        $t_{i+1,j}(X) \leftarrow t_{i,2j-1}(X) \cdot t_{i,2j}(X)$   $\triangleright$  Using fast polynomial multiplication
6:     end for
7:   end for
8:   return  $h = t_{\log_2(d),1}$ 
9: end procedure

```

---

In round  $i$  there are  $d/2^i$  polynomial multiplications for polynomials of degree  $2^{i-1}$ . In FFT-friendly<sup>3</sup> fields, polynomial multiplication can be per-

---

<sup>3</sup> These are fields where there exists an element that has a smooth order of at least  $d$ .

formed in time  $O(d \log(d))$ .<sup>4</sup> The total running time of the algorithm is therefore  $\sum_{i=1}^{\log_2(d)} \frac{d}{2^i} 2^{i-1} \log(2^{i-1}) = \sum_{i=1}^{\log_2(d)} O(d \cdot i) = O(d \log^2(d))$ .

Algorithm 2 naturally extends to more complicated, low-depth circuits. Addition gates are performed directly through polynomial addition, which takes  $O(d)$  time for degree  $d$  polynomials. As long as the circuit is low-depth and has  $O(d)$  multiplication gates, the complexity remains  $O(d \log^2(d))$ . Furthermore, we can compute  $r^k(X)$  for  $k \leq d$  using only a single FFT of length  $\deg(r) \cdot k$  for an input polynomial  $r$ . The FFT evaluates  $r$  at  $\deg(r) \cdot k$  points. Then we raise each point to the power of  $k$ . This takes time  $O(\deg(r) \cdot k(\log(\deg(r)) + \log(k)))$  and saves a factor of  $\log(k)$  over a repeated squaring implementation.

*Batching.* Multiple sumcheck instances, e.g.  $(s, [[f]])$  and  $(s', [[g]])$  can easily be batched together. This is done using a random-linear combination, i.e. showing that  $(s + \alpha s', [[f]] + \alpha [[g]]) \in \mathcal{L}(\mathcal{R}_{\text{SUM}})$  for a random verifier-generated  $\alpha$  [23, 48]. The batching step has soundness  $\frac{1}{\mathbb{F}}$ .

### 3.2 ZeroCheck PIOP

In this section, we describe a PIOP showing that a multivariate polynomial evaluates to zero everywhere on the boolean hypercube. The PIOP builds upon the sumcheck PIOP in Sect. 3.1 and is a key building block for product-check PIOP in Sect. 3.3. The zerocheck PIOP is also helpful in HyperPlonk for proving the gate identity.

**Definition 3.3 (ZeroCheck relation).** *The relation  $\mathcal{R}_{\text{ZERO}}$  is the set of all tuples  $(\mathbf{x}; \mathbf{w}) = (([[f]]); f)$  where  $f \in \mathcal{F}_\mu^{(\leq d)}$  and  $f(\mathbf{x}) = 0$  for all  $\mathbf{x} \in B_\mu$ .*

We use an idea from [43] to reduce a ZeroCheck to a SumCheck.

*Construction.* Given a tuple  $(\mathbf{x}; \mathbf{w}) = (([[f]]); f)$ , the protocol is the following:

- $\mathcal{V}$  sends  $\mathcal{P}$  a random vector  $\mathbf{r} \xleftarrow{\$} \mathbb{F}^\mu$
- Let  $\hat{f}(\mathbf{X}) := f(\mathbf{X}) \cdot eq(\mathbf{X}, \mathbf{r})$  where  $eq(\mathbf{x}, \mathbf{y}) := \prod_{i=1}^\mu (x_i y_i + (1 - x_i)(1 - y_i))$ .
- Run a sumcheck PolyIOP to convince the verifier that  $((0, [[\hat{f}]]); \hat{f}) \in \mathcal{R}_{\text{SUM}}$ .

*Batching.* It is possible to batch two instances  $(([[f]]); f) \in \mathcal{R}_{\text{ZERO}}$  and  $(([[g]]); g) \in \mathcal{R}_{\text{ZERO}}$  by running a zerocheck on  $(([[f + \alpha g]]); f + \alpha g)$  for a random  $\alpha \in \mathbb{F}$ . The soundness error of the batching protocol  $\frac{1}{\mathbb{F}}$ .

**Theorem 3.4.** *The PIOP for  $\mathcal{R}_{\text{ZERO}}$  is perfectly complete and has knowledge error  $\delta_{\text{zero}}^{d, \mu} := d\mu/|\mathbb{F}| + \delta_{\text{sum}}^{d+1, \mu} = O(d\mu/|\mathbb{F}|)$ .*

<sup>4</sup> Recent breakthrough results have shown that polynomial multiplication is  $O(d \log(d))$  over arbitrary finite fields [35] and there have been efforts toward building practical, fast multiplication algorithms for arbitrary fields [9]. In practice, and especially for low-degree polynomials, using Karatsuba multiplication might be faster.

### 3.3 ProductCheck PIOP

We describe a PIOP for the product check relation, that is, for a rational polynomial (where both the nominator and the denominator are multivariate polynomials), the product of the evaluations on the boolean hypercube is a claimed value  $s$ . The PIOP uses the idea from the Quark system [44, §5], we adapt it to build upon the zerocheck PIOP in Sect. 3.2. Product check PIOP is a key building block for the multiset equality check PIOP in Sect. 3.4.

**Definition 3.5 (ProductCheck relation).** *The relation  $\mathcal{R}_{PROD}$  is the set of all tuples  $(\mathbf{x}; \mathbf{w}) = ((s, [[f_1]], [[f_2]]); f_1, f_2)$  where  $f_1 \in \mathcal{F}_\mu^{(\leq d)}$ ,  $f_2 \in \mathcal{F}_\mu^{(\leq d)}$ ,  $f_2(b) \neq 0 \forall b \in B_\mu$  and  $\prod_{\mathbf{x} \in B_\mu} f'(\mathbf{x}) = s$ , where  $f'$  is the rational polynomial  $f' := f_1/f_2$ . In the case that  $f_2 = c$  is a constant polynomial, we directly set  $f := f_1/c$  and write  $(\mathbf{x}; \mathbf{w}) = ((s, [[f]]); f)$ .*

*Construction.* The Quark system [44, §5] constructs a proof system for the  $\mathcal{R}_{PROD}$  relation. The proof system uses an instance of the  $\mathcal{R}_{ZERO}$  PolyIOP on  $\mu + 1$  variables. Given a tuple  $(\mathbf{x}; \mathbf{w}) = ((s, [[f_1]], [[f_2]]); f_1, f_2)$ , we denote by  $f' := f_1/f_2$ . The protocol is the following:

- $\mathcal{P}$  sends an oracle  $\tilde{v} \in \mathcal{F}_{\mu+1}^{(\leq 1)}$  such that for all  $\mathbf{x} \in B_\mu$ ,

$$\tilde{v}(0, \mathbf{x}) = f'(\mathbf{x}), \quad \tilde{v}(1, \mathbf{x}) = \tilde{v}(\mathbf{x}, 0) \cdot \tilde{v}(\mathbf{x}, 1), \quad \tilde{v}(1) = 0.$$

- Define  $\hat{h} := \text{merge}(\hat{f}, \hat{g}) \in \mathcal{F}_{\mu+1}^{(\leq \max(2, d+1))}$  where

$$\hat{f}(\mathbf{X}) := \tilde{v}(1, \mathbf{X}) - \tilde{v}(\mathbf{X}, 0) \cdot \tilde{v}(\mathbf{X}, 1), \quad \hat{g}(\mathbf{X}) := f_2(\mathbf{X}) \cdot \tilde{v}(0, \mathbf{X}) - f_1(\mathbf{X}).$$

Run a ZeroCheck PolyIOP for  $([[\hat{h}]]; \hat{h}) \in \mathcal{R}_{ZERO}$ , i.e., the polynomial  $\tilde{v}$  is computed correctly.

- $\mathcal{V}$  queries  $[[\tilde{v}]]$  at point  $(1, \dots, 1, 0) \in \mathbb{F}^{\mu+1}$ , and checks that the evaluation is  $s$ .

**Theorem 3.6.** *Let  $d' := \max(2, d+1)$ . The PIOP for  $\mathcal{R}_{PROD}$  is perfectly complete and has knowledge error  $\delta_{prod}^{d, \mu} := \delta_{zero}^{d', \mu+1} = \mathcal{O}(d' \mu / |\mathbb{F}|)$ .*

### 3.4 Multiset Check PIOP

We describe a multivariate PIOP checking that two multisets are equal. The PIOP builds upon the product-check PIOP in Sect. 3.3. The multiset check PIOP is a key building block for the permutation PIOP in Sect. 3.5 and the lookup PIOP in Sect. 3.6. A similar idea has been proposed in the univariate polynomial setting by Gabizon in a blogpost [26].

**Definition 3.7 (Multiset Check relation).** *For any  $k \geq 1$ , the relation  $\mathcal{R}_{MSET}^k$  is the set of all tuples  $(\mathbf{x}; \mathbf{w}) = ((([f_1]], \dots, [[f_k]], [[g_1]], \dots, [[g_k]]); (f_1, \dots, f_k, g_1, \dots, g_k))$  where  $f_i, g_i \in \mathcal{F}_\mu^{(\leq d)}$  ( $1 \leq i \leq k$ ) and the following two multisets of tuples are equal:  $\left\{ \mathbf{f}_\mathbf{x} := [f_1(\mathbf{x}), \dots, f_k(\mathbf{x})] \right\}_{\mathbf{x} \in B_\mu} = \left\{ \mathbf{g}_\mathbf{x} := [g_1(\mathbf{x}), \dots, g_k(\mathbf{x})] \right\}_{\mathbf{x} \in B_\mu}$ .*



*Basic Construction.* We start by describing a PolyIOP for  $\mathcal{R}_{\text{MSET}}^1$ . The protocol can be obtained from a protocol for  $\mathcal{R}_{\text{PROD}}$ . Given a tuple  $(([[f]], [[g]]); (f, g))$ , the protocol is the following:

- $\mathcal{V}$  samples and sends  $\mathcal{P}$  a challenge  $r \xleftarrow{\$} \mathbb{F}$ .
- Set  $f' := r + f$  and  $g' := r + g$
- If  $g' \neq 0 \forall b \in B_\mu$  run a ProductCheck PolyIOP for  $((1, [[f']], [[g']]); (f', g') \in \mathcal{R}_{\text{PROD}}$ .
- Else the prover sends  $b$  such that  $g'(b) = 0$  and the verifier accepts if  $g(b) = -r$  (this case happens with negligible probability).

**Theorem 3.8.** *The PIOP for  $\mathcal{R}_{\text{MSET}}^1$  has perfect completeness and has knowledge error  $\delta_{\text{mset},1}^{d,\mu} := 2^{\mu+1}/|\mathbb{F}| + \delta_{\text{prod}}^{d,\mu} = \mathcal{O}((2^\mu + d\mu)/|\mathbb{F}|)$ .*

### 3.5 Permutation PIOP

We describe a multivariate PIOP showing that for two multivariate polynomials  $f, g \in \mathcal{F}_\mu^{(\leq d)}$ , the evaluations of  $g$  on the boolean hypercube is a predefined permutation  $\sigma$  of  $f$ 's evaluations on the boolean hypercube. The permutation PIOP is a key building block of HyperPlonk for proving the wiring identity.

**Definition 3.9 (Permutation relation).** *The indexed relation  $\mathcal{R}_{\text{PERM}}$  is the set of tuples  $(\mathbf{i}; \mathbf{x}; \mathbf{w}) = (\sigma; ([[f]], [[g]]); (f, g))$ , where  $\sigma : B_\mu \rightarrow B_\mu$  is a permutation,  $f, g \in \mathcal{F}_\mu^{(\leq d)}$ , and  $g(\mathbf{x}) = f(\sigma(\mathbf{x}))$  for all  $\mathbf{x} \in B_\mu$ .*

*Construction.* Gabizon et al. [29] construct a permutation argument. We adapt their scheme into a multivariate PolyIOP. The construction uses a PolyIOP instance for  $\mathcal{R}_{\text{MSET}}$ . Given a tuple  $(\sigma; ([[f]], [[g]]); (f, g))$  where  $\sigma$  is the predefined permutation, the indexer generates two oracles  $[[s_{\text{id}}]], [[s_\sigma]]$  such that  $s_{\text{id}} \in \mathcal{F}_\mu^{(\leq 1)}$  maps each  $\mathbf{x} \in B_\mu$  to  $[\mathbf{x}] := \sum_{i=1}^\mu \mathbf{x}_i \cdot 2^{i-1} \in \mathbb{F}$ , and  $s_\sigma \in \mathcal{F}_\mu^{(\leq 1)}$  maps each  $\mathbf{x} \in B_\mu$  to  $[\sigma(\mathbf{x})]$ .<sup>5</sup> The PolyIOP is the following:

- Run a Multiset Check PolyIOP for  $(([[s_{\text{id}}]], [[f]], [[s_\sigma]], [[g]]); (s_{\text{id}}, f, s_\sigma, g)) \in \mathcal{R}_{\text{MSET}}^2$ .

**Theorem 3.10.** *The PIOP for  $\mathcal{R}_{\text{PERM}}$  is perfectly complete and has knowledge error  $\delta_{\text{perm}}^{d,\mu} := \delta_{\text{mset},2}^{d,\mu} = \mathcal{O}((2^\mu + d\mu)/|\mathbb{F}|)$ .*

### 3.6 Lookup PIOP

This section describes a multivariate PIOP checking the table lookup relation. The PIOP builds upon the multiset check PIOP (Sect. 3.4) and is a key building block for HyperPlonk+. Our construction is inspired by a univariate PIOP for the table lookup relation called Plookup [27]. However, it is non-trivial to adapt Plookup to the multivariate setting because their scheme requires the existence

<sup>5</sup> Here we further require  $|\mathbb{F}| \geq 2^\mu$  so that  $[\mathbf{x}]$  never overflow.

of a subdomain of the polynomial that is a cyclic subgroup  $\mathbb{G}$  with a generator  $\omega \in \mathbb{G}$ . Translating to the multilinear case, we need to build an efficient function  $g$  that generates the entire boolean hypercube; moreover,  $g$  has to be linear so that the degree of the polynomial does not blow up. However, such a linear function does not exist. Fortunately, we can construct a quadratic function from  $\mathbb{F}^\mu$  to  $\mathbb{F}^\mu$  that traverses  $B_\mu$ . We then show how to linearize it by modifying some of the building blocks that Plookup uses. This gives an efficient Plookup protocol over the hypercube.

**Definition 3.11 (Lookup relation).** *The indexed relation  $\mathcal{R}_{\text{LOOKUP}}$  is the set of tuples  $(i; \mathbf{x}; \mathbf{w}) = (\mathbf{t}; [[f]]; (f, \text{addr}))$  where  $\mathbf{t} \in \mathbb{F}^{2^\mu - 1}$ ,  $f \in \mathcal{F}_\mu^{(\leq d)}$ , and  $\text{addr} : B_\mu \rightarrow [1, 2^\mu]$  is a map such that  $f(\mathbf{x}) = \mathbf{t}_{\text{addr}(\mathbf{x})}$  for all  $\mathbf{x} \in B_\mu$ .*

Before presenting the PIOP for  $\mathcal{R}_{\text{LOOKUP}}$ , we first show how to build a quadratic function that generates the entire boolean hypercube.

*A Quadratic Generator in  $\mathbb{F}_{2^\mu}$ .* For every  $\mu \in \mathbb{N}$ , we fix a primitive polynomial  $p_\mu \in \mathbb{F}_2[X]$  where  $p_\mu := X^\mu + \sum_{s \in S} X^s + 1$  for some set  $S \subseteq [\mu - 1]$ , so that  $\mathbb{F}_2[X]/(p_\mu) \cong \mathbb{F}_2^\mu[X] \cong \mathbb{F}_{2^\mu}$ . By definition of primitive polynomials,  $X \in \mathbb{F}_2^\mu[X]$  is a generator of  $\mathbb{F}_2^\mu[X] \setminus \{0\}$ . This naturally defines a generator function  $g_\mu : B_\mu \rightarrow B_\mu$  as  $g_\mu(\mathbf{b}_1, \dots, \mathbf{b}_\mu) = (\mathbf{b}_\mu, \mathbf{b}'_1, \dots, \mathbf{b}'_{\mu-1})$ , where  $\mathbf{b}'_i = \mathbf{b}_i \oplus \mathbf{b}_\mu$  ( $i \leq 1 < \mu$ ) if  $i \in S$ , and  $\mathbf{b}'_i = \mathbf{b}_i$  otherwise. Essentially, for a polynomial  $f \in \mathbb{F}_2^\mu[X]$  with coefficients  $\mathbf{b}$ ,  $g_\mu(\mathbf{b})$  is the coefficient vector of  $X \cdot f(X)$ . Hence the following lemma is straightforward.

**Lemma 3.12.** *Let  $g_\mu : B_\mu \rightarrow B_\mu$  be the generator function defined above. For every  $\mathbf{x} \in B_\mu \setminus \{0^\mu\}$ , it holds that  $\{g_\mu^{(i)}(\mathbf{x})\}_{i \in [2^\mu - 1]} = B_\mu \setminus \{0^\mu\}$ , where  $g_\mu^{(i)}(\cdot)$  denotes  $i$  repeated application of  $g_\mu$ .*

Directly composing a polynomial  $f$  with the generator  $g$  will blow up the degree of the resulting polynomial; moreover, the prover needs to send the composed oracle  $f(g(\cdot))$ . Both of which affect the efficiency of the PIOP. We address the issue by describing a trick that manipulates  $f$  in a way that simulates the behavior of  $f(g(\cdot))$  on the boolean hypercube, but without blowing up the degree.

*Linearizing the Generator.* For a multivariate polynomial  $f \in \mathcal{F}_\mu^{(\leq d)}$ , we define  $f_{\Delta_\mu} \in \mathcal{F}_\mu^{(\leq d)}$  as

$$f_{\Delta_\mu}(\mathbf{X}_1, \dots, \mathbf{X}_\mu) := \mathbf{X}_\mu \cdot f(1, \mathbf{X}'_1, \dots, \mathbf{X}'_{\mu-1}) + (1 - \mathbf{X}_\mu) \cdot f(0, \mathbf{X}_1, \dots, \mathbf{X}_{\mu-1})$$

where  $\mathbf{X}'_i := 1 - \mathbf{X}_i$  ( $i \leq 1 < \mu$ ) if  $i \in S$ , and  $\mathbf{X}'_i := \mathbf{X}_i$  otherwise.

**Lemma 3.13.** *For every  $\mu \in \mathbb{N}$ , let  $g_\mu : B_\mu \rightarrow B_\mu$  be the generator function defined in Lemma 3.12. For every  $d \in \mathbb{N}$  and polynomial  $f \in \mathcal{F}_\mu^{(\leq d)}$ , it holds that  $f_{\Delta_\mu}(\mathbf{x}) = f(g_\mu(\mathbf{x}))$  for every  $\mathbf{x} \in B_\mu$ . Moreover,  $f_{\Delta_\mu}$  has individual degree  $d$  and one can evaluate  $f_{\Delta_\mu}$  from 2 evaluations of  $f$ .*

*Proof.* By definition,  $f_{\Delta_\mu}$  has individual degree  $d$  and an evaluation of  $f_{\Delta_\mu}$  can be derived from 2 evaluations of  $f$ . Next, we argue that  $f_{\Delta_\mu}(\mathbf{x}) = f(g_\mu(\mathbf{x}))$  for every  $\mathbf{x} \in B_\mu$ .

First,  $f_{\Delta_\mu}(0^\mu) = f(g_\mu(0^\mu))$  because  $f_{\Delta_\mu}(0^\mu) = f(0^\mu)$  and  $g_\mu(0^\mu) = 0^\mu$  by definition of  $f_{\Delta_\mu}, g_\mu$ . Second, for every  $\mathbf{x} \in B_\mu \setminus \{0^\mu\}$ , by definition of  $g_\mu$ ,

$$f(g_\mu(\mathbf{x}_1, \dots, \mathbf{x}_\mu)) = f(\mathbf{x}_\mu, \mathbf{x}'_1, \dots, \mathbf{x}'_{\mu-1}),$$

where  $\mathbf{x}'_i = \mathbf{x}_i \oplus \mathbf{x}_\mu$  ( $i \leq 1 < \mu$ ) for every  $i$  in the fixed set  $S$ , and  $\mathbf{x}'_i = \mathbf{x}_i$  otherwise. We observe that  $\mathbf{x}_i \oplus \mathbf{x}_\mu = 1 - \mathbf{x}_i$  when  $\mathbf{x}_\mu = 1$  and  $\mathbf{x}_i \oplus \mathbf{x}_\mu = \mathbf{x}_i$  when  $\mathbf{x}_\mu = 0$ , thus we can rewrite

$$\begin{aligned} f(\mathbf{x}_\mu, \mathbf{x}'_1, \dots, \mathbf{x}'_{\mu-1}) &= \mathbf{x}_\mu \cdot f(1, \mathbf{x}_1^*, \dots, \mathbf{x}_{\mu-1}^*) + (1 - \mathbf{x}_\mu) \cdot f(0, \mathbf{x}_1, \dots, \mathbf{x}_{\mu-1}) \\ &= f_{\Delta_\mu}(\mathbf{x}_1, \dots, \mathbf{x}_\mu) \end{aligned}$$

where  $\mathbf{x}_i^* = 1 - \mathbf{x}_i$  ( $i \leq 1 < \mu$ ) for every  $i$  in the fixed set  $S$ , and  $\mathbf{x}_i^* = \mathbf{x}_i$  otherwise. The last equality holds by definition of  $f_{\Delta_\mu}$ . In summary,  $f(g_\mu(\mathbf{x}_1, \dots, \mathbf{x}_\mu)) = f_{\Delta_\mu}(\mathbf{x}_1, \dots, \mathbf{x}_\mu)$  for every  $B_\mu$  and the lemma holds.

*Construction.* Now we are ready to present the PIOP for  $\mathcal{R}_{\text{LOOKUP}}$ , which is an adaptation of Plookup [27] in the multivariate setting. The PIOP invokes a protocol for  $\mathcal{R}_{\text{MSET}}^2$ . We introduce a notation that embeds a vector to the hypercube while still preserving the vector order with respect to the generator function. For a vector  $\mathbf{t} \in \mathbb{F}^{2^\mu - 1}$ , we denote by  $t \leftarrow \text{emb}(\mathbf{t}) \in \mathcal{F}_\mu^{(\leq 1)}$  the multilinear polynomial such that  $t(0^\mu) = 0$  and  $t(g_\mu^{(i)}(1, 0^{\mu-1})) = \mathbf{t}_i$  for every  $i \in [2^\mu - 1]$ . By Lemma 3.12,  $t$  is well-defined and embeds the entire vector  $\mathbf{t}$  onto  $B_\mu \setminus \{0^\mu\}$ .

For an index  $\mathbf{t} \in \mathbb{F}^{2^\mu - 1}$ , the indexer generates an oracle  $[[t]]$  where  $t \leftarrow \text{emb}(\mathbf{t})$ . For a tuple  $(\mathbf{t}; [[f]]; (f, \text{addr}))$  where  $f(B_\mu) \subseteq t(B_\mu) \setminus \{0\}$ , let  $(\mathbf{a}_1, \dots, \mathbf{a}_{2^\mu - 1})$  be the vector where  $\mathbf{a}_i \in \mathbb{N}$  is the number of appearance of  $\mathbf{t}_i$  in  $f(B_\mu)$ . Note that  $\sum_{i=1}^{2^\mu - 1} \mathbf{a}_i = 2^\mu$ . Denote by  $\mathbf{h} \in \mathbb{F}^{2^{\mu+1} - 1}$  the vector

$$\mathbf{h} := \underbrace{(\mathbf{t}_1, \dots, \mathbf{t}_1)}_{1+\mathbf{a}_1}, \mathbf{t}_2, \dots, \mathbf{t}_{i-1}, \underbrace{(\mathbf{t}_i, \dots, \mathbf{t}_i)}_{1+\mathbf{a}_i}, \mathbf{t}_{i+1}, \dots, \mathbf{t}_{2^\mu - 2}, \underbrace{(\mathbf{t}_{2^\mu - 1}, \dots, \mathbf{t}_{2^\mu - 1})}_{1+\mathbf{a}_{2^\mu - 1}}.$$

We present the protocol below:

- $\mathcal{P}$  sends  $\mathcal{V}$  oracles  $[[h]]$ , where  $h \leftarrow \text{emb}(\mathbf{h}) \in \mathcal{F}_{\mu+1}^{(\leq 1)}$ .
- Define  $g_1 := \text{merge}(f, t) \in \mathcal{F}_{\mu+1}^{(\leq d)}$  and  $g_2 := \text{merge}(f, t_{\Delta_\mu}) \in \mathcal{F}_{\mu+1}^{(\leq d)}$ , where  $\text{merge}$  is defined in Eq. (7). Run a multiset check PIOP (Sect. 3.4) for  $(([[g_1]], [[g_2]], [[h]], [[h_{\Delta_{\mu+1}}]]); (f, t, h)) \in \mathcal{R}_{\text{MSET}}^2$ .
- $\mathcal{V}$  queries  $h(0^{\mu+1})$  and checks that the answer equals 0.

**Theorem 3.14.** *The PIOP for  $\mathcal{R}_{\text{LOOKUP}}$  is perfectly complete and has knowledge error  $\delta_{\text{lookup}}^{d, \mu} := \delta_{\text{mset}, 2}^{d, \mu+1} = \mathcal{O}((2^\mu + d\mu)/|\mathbb{F}|)$ .*

### 3.7 Batch Openings

This section describes a batching protocol proving the correctness of multiple multivariate polynomial evaluations. Essentially, the protocol reduces multiple oracle queries to different polynomials into a *single* query to a multivariate oracle. The batching protocol is helpful for HyperPlonk to enable efficient batch evaluation openings. In particular, the SNARK prover only needs to compute a single multilinear PCS evaluation proof, even if there are multiple PCS evaluations.

We note that Thaler [47, §4.5.2] shows how to batch two evaluations of a *single* multilinear polynomial. The algorithm can be generalized for multiple evaluations of *different* multilinear polynomials. However, the prover time complexity is  $O(k^2 \mu \cdot 2^\mu)$  where  $k$  is the number of evaluations, and  $\mu$  is the number of variables. In comparison, our algorithm achieves complexity  $O(k \cdot 2^\mu)$  which is  $k\mu$ -factor faster. Note that  $O(k \cdot 2^\mu)$  is already optimal as the prover needs to take  $O(k \cdot 2^\mu)$  time to evaluate  $\{f_i(\mathbf{z}_i)\}_{i \in [k]}$  before batching.

**Definition 3.15 (BatchEval relation).** *The relation  $\mathcal{R}_{\text{BATCH}}^k$  is the set of all tuples  $(\mathfrak{x}; \mathfrak{w}) = ((\mathbf{z}_i)_{i \in [k]}, (y_i)_{i \in [k]}, ([[f_i]])_{i \in [k]}; (f_i)_{i \in [k]})$  where  $\mathbf{z}_i \in \mathbb{F}^\mu$ ,  $y_i \in \mathbb{F}$ ,  $f_i \in \mathcal{F}_\mu^{(\leq d)}$  and  $f_i(\mathbf{z}_i) = y_i$  for all  $i \in [k]$ .*

*Remark 3.16.* The polynomials  $\{f_i\}_{i \in [k]}$  are all  $\mu$ -variate. This is without loss of generality. E.g., suppose one of the evaluated polynomial  $f'_j$  has only  $\mu - 1$  variables, we can define  $f_j(Y, \mathbf{X}) = Y \cdot f'_j(\mathbf{X}) + (1 - Y) \cdot f'_j(\mathbf{X})$  which is essentially  $f'_j$  but with  $\mu$  variables. The same trick easily extends to  $f'_j$  with arbitrary  $\mu' < \mu$  variables.

*Construction.* For ease of exposition, we consider the case where  $f_1, \dots, f_k$  are *multilinear*. We emphasize that the same techniques can be extended for multivariate polynomials.

Assume w.l.o.g that  $k = 2^\ell$  is a power of 2. We observe that  $\mathcal{R}_{\text{BATCH}}^k$  is essentially a ZeroCheck relation over the set  $Z := \{\mathbf{z}_i\}_{i \in [k]} \subseteq \mathbb{F}^\mu$ , that is, for every  $i \in [k]$ ,  $f_i(\mathbf{z}_i) - y_i = 0$ . Nonetheless,  $Z$  is outside the boolean hypercube, and we cannot directly reuse the ZeroCheck PIOP.

The key idea is to interpret each zero constraint as a sumcheck via multilinear extension, so that we can work on the boolean hypercube later. In particular, for every  $i \in [k]$ , we want to constrain  $f_i(\mathbf{z}_i) - y_i = 0$ . Since  $f_i$  is multilinear, by definition of multilinear extension, this is equivalent to constraining that

$$c_i := \left( \sum_{\mathbf{b} \in B_\mu} f_i(\mathbf{b}) \cdot eq(\mathbf{b}, \mathbf{z}_i) \right) - y_i = 0. \tag{8}$$

Note that Eq. (8) holds for every  $i \in [k]$  if and only if the polynomial  $\sum_{i \in [k]} eq(\mathbf{Z}, \langle i \rangle) \cdot c_i$  is identically zero, where  $\langle i \rangle$  is  $\ell$ -bit representation of  $i - 1$ . By the Schwartz Zippel Lemma, it is sufficient to check that for a random

vector  $\mathbf{t} \stackrel{s}{\leftarrow} \mathbb{F}^\ell$ , it holds that

$$\sum_{i \in [k]} eq(\mathbf{t}, \langle i \rangle) \cdot c_i = \sum_{i \in [k]} eq(\mathbf{t}, \langle i \rangle) \cdot \left[ \left( \sum_{\mathbf{b} \in B_\mu} f_i(\mathbf{b}) \cdot eq(\mathbf{b}, \mathbf{z}_i) \right) - y_i \right] = 0. \quad (9)$$

Next, we arithmetize equation (9) and make it an algebraic formula. For every  $(i, \mathbf{b}) \in [k] \times B_\mu$ , we set value  $g_{i,\mathbf{b}} := eq(\mathbf{t}, \langle i \rangle) \cdot f_i(\mathbf{b})$ , and define an MLE  $\tilde{g}$  for  $(g_{i,\mathbf{b}})_{i \in [k], \mathbf{b} \in B_\mu}$  such that  $\tilde{g}(\langle i \rangle, \mathbf{b}) = g_{i,\mathbf{b}} \forall (i, \mathbf{b}) \in [k] \times B_\mu$ ; similarly, we define an MLE  $\tilde{eq}$  for  $(eq(\mathbf{b}, \mathbf{z}_i))_{i \in [k], \mathbf{b} \in B_\mu}$  where  $\tilde{eq}(\langle i \rangle, \mathbf{b}) = eq(\mathbf{b}, \mathbf{z}_i) \forall (i, \mathbf{b}) \in [k] \times B_\mu$ . Let  $s := \sum_{i \in [k]} eq(\mathbf{t}, \langle i \rangle) \cdot y_i$ , then Eq. (9) can be rewritten as

$$\sum_{i \in [k], \mathbf{b} \in B_\mu} \tilde{g}(\langle i \rangle, \mathbf{b}) \cdot \tilde{eq}(\langle i \rangle, \mathbf{b}) = s.$$

This is equivalent to prove a sumcheck claim for the degree-2 polynomial  $g^* := \tilde{g}(\mathbf{Y}, \mathbf{X}) \cdot \tilde{eq}(\mathbf{Y}, \mathbf{X})$  over set  $B_{\ell+\mu}$ . Hence we obtain the following PIOP protocol in Algorithm 3. Note that  $g^*$  is only with degree 2. Thus we can run a classic sumcheck without sending any univariate oracles.

---

**Algorithm 3.** Batch evaluation of multi-linear polynomials

---

- 1: **procedure** `BATCHEVAL` ( $[f_i \in \mathcal{F}_\mu^{(\leq 1)}, \mathbf{z}_i \in \mathbb{F}^\mu, y_i \in \mathbb{F}]_{i=1}^k$ )
  - 2:    $\mathcal{V}$  sends  $\mathcal{P}$  a random vector  $\mathbf{t} \stackrel{s}{\leftarrow} \mathbb{F}^\ell$ .
  - 3:   Define sum  $s := \sum_{i \in [k]} eq(\mathbf{t}, \langle i \rangle) \cdot y_i$ .
  - 4:   Let  $\tilde{g}$  be the MLE for  $(g_{i,\mathbf{b}})_{i \in [k], \mathbf{b} \in B_\mu}$  where  $g_{i,\mathbf{b}} := eq(\mathbf{t}, \langle i \rangle) \cdot f_i(\mathbf{b})$ .
  - 5:   Let  $\tilde{eq}$  be the MLE for  $(eq(\mathbf{b}, \mathbf{z}_i))_{i \in [k], \mathbf{b} \in B_\mu}$  such that  $\tilde{eq}(\langle i \rangle, \mathbf{b}) = eq(\mathbf{b}, \mathbf{z}_i)$ .
  - 6:    $\mathcal{P}$  and  $\mathcal{V}$  run a SumCheck PIOP for  $(s, [[g^*]]; g^*) \in \mathcal{R}_{\text{SUM}}$ , where  $g^* := \tilde{g} \cdot \tilde{eq}$ .
  - 7:   Let  $(\mathbf{a}_1, \mathbf{a}_2) \in \mathbb{F}^{\ell+\mu}$  be the sumcheck challenge vector.  $\mathcal{P}$  answers the oracle query  $\tilde{g}(\mathbf{a}_1, \mathbf{a}_2)$ .
  - 8:    $\mathcal{V}$  evaluates  $\tilde{eq}(\mathbf{a}_1, \mathbf{a}_2)$  herself, and checks that  $\tilde{g}(\mathbf{a}_1, \mathbf{a}_2) \cdot \tilde{eq}(\mathbf{a}_1, \mathbf{a}_2)$  is consistent with the last message of the sumcheck.
  - 9: **end procedure**
- 

*Remark 3.17.* If the SNARK is using a homomorphic commitment scheme, to answer query  $\tilde{g}(\mathbf{a}_1, \mathbf{a}_2)$  the prover only needs to provide a single PCS opening proof for a  $\mu$ -variate polynomial  $g'(\mathbf{X}) := \tilde{g}(\mathbf{a}_1, \mathbf{X}) = \sum_{i \in [k]} eq(\langle i \rangle, \mathbf{a}_1) \cdot eq(\mathbf{t}, \langle i \rangle) \cdot f_i(\mathbf{X})$  on point  $\mathbf{a}_2$ . The verifier can evaluate  $\{eq(\langle i \rangle, \mathbf{a}_1) \cdot eq(\mathbf{t}, \langle i \rangle)\}_{i \in [k]}$  in time  $O(k)$ , and homomorphically compute  $g'$ 's commitment from the commitments to  $\{f_i\}_{i \in [k]}$ , and checks the opening proof against  $g'$ 's commitment. Finally, the verifier checks that  $g'(\mathbf{a}_2)$  matches the claimed evaluation  $\tilde{g}(\mathbf{a}_1, \mathbf{a}_2)$ .

*Analysis.* The PIOP for  $\mathcal{R}_{\text{BATCH}}$  is complete and knowledge-sound given the completeness and knowledge-soundness of the sumcheck PIOP.

Next, we analyze the complexity of the protocol: The prover time is  $O(k \cdot 2^\mu)$  as it runs a sumcheck PIOP for a polynomial  $g^* := \tilde{g} \cdot \tilde{e}q$  of degree 2 and  $\mu + \log k$  variables, where  $\tilde{g}$  and  $\tilde{e}q$  can both be constructed in time  $O(k \cdot 2^\mu)$ . Note that this is already optimal as the prover anyway needs to take  $O(k \cdot 2^\mu)$  time to evaluate  $\{f_i(\mathbf{z}_i)\}_{i \in [k]}$  before batching. The verifier takes time  $O(\mu + \log k)$  in the sumcheck; the sum  $s$  can be computed in time  $O(k)$ ; the evaluation  $\tilde{e}q(\mathbf{a}_1, \mathbf{a}_2) = \sum_{i \in [k]} eq(\mathbf{a}_1, \langle i \rangle) \cdot \tilde{e}q(\langle i \rangle, \mathbf{a}_2)$  can be derived from  $\mathbf{a}_1$  and the  $k$  evaluations  $\{\tilde{e}q(\langle i \rangle, \mathbf{a}_2) = eq(\mathbf{a}_2, \mathbf{z}_i)\}_{i \in [k]}$  where each evaluation  $eq(\mathbf{a}_2, \mathbf{z}_i)$  takes time  $O(\mu)$ . In summary, the verifier time is  $O(k\mu)$ .

**A More Efficient Batching Scheme in a Special Setting.** Sometimes one only needs to open a *single* multilinear polynomial at multiple points, where each point is *in the boolean hypercube*. In this setting, we provide a more efficient algorithm with complexity  $O(2^\mu)$  which is  $k$  times faster than Algorithm 3. We also note that the technique can be used to construct an efficient Commit-and-Prove SNARK scheme from multilinear commitments.

## 4 HyperPlonk: Plonk on the Boolean Hypercube

Equipped with the building blocks in Sect. 3, we now describe the Polynomial IOP for HyperPlonk. In Sect. 4.1, we introduce  $\mathcal{R}_{\text{PLONK}}$  — an indexed relation on the boolean hypercube that generalizes the vanilla Plonk constraint system [29]. We present a Polynomial IOP protocol for  $\mathcal{R}_{\text{PLONK}}$  and analyze its security and efficiency in Sect. 4.2.

### 4.1 Constraint Systems

*Notation.* For any  $m \in \mathbb{Z}$  and  $i \in [0, 2^m)$ , we use  $\langle i \rangle_m = \mathbf{v} \in B_m$  to denote the  $m$ -bit binary representation of  $i$ , that is,  $i = \sum_{j=1}^m \mathbf{v}_j \cdot 2^{j-1}$ .

**Definition 4.1 (HyperPlonk indexed relation).** Fix public parameters  $\text{gp} := (\mathbb{F}, \ell, n, \ell_w, \ell_q, f)$  where  $\mathbb{F}$  is the field,  $\ell = 2^\nu$  is the public input length,  $n = 2^\mu$  is the number of constraints,  $\ell_w = 2^{\nu_w}$ ,  $\ell_q = 2^{\nu_q}$  are the number of witnesses and selectors per constraint<sup>6</sup>, and  $f : \mathbb{F}^{\ell_q + \ell_w} \rightarrow \mathbb{F}$  is an algebraic map with degree  $d$ . The indexed relation  $\mathcal{R}_{\text{PLONK}}$  is the set of all tuples

$$(\mathbf{i}; \mathbf{x}; \mathbf{w}) = ((q, \sigma); (p, [[w]]); w),$$

where  $\sigma : B_{\mu+\nu_w} \rightarrow B_{\mu+\nu_w}$  is a permutation,  $q \in \mathcal{F}_{\mu+\nu_q}^{(\leq 1)}$ ,  $p \in \mathcal{F}_{\mu+\nu}^{(\leq 1)}$ ,  $w \in \mathcal{F}_{\mu+\nu_w}^{(\leq 1)}$ , such that

---

<sup>6</sup> We can pad zeroes if the actual number is not a power of two.

- the wiring identity is satisfied, that is,  $(\sigma; ([[w]], [[w]]); w) \in \mathcal{R}_{PERM}$  (Definition 3.9);
- the gate identity is satisfied, that is,  $([[\tilde{f}]]); \tilde{f}) \in \mathcal{R}_{ZERO}$  (Definition 3.3), where the virtual polynomial  $\tilde{f} \in \mathcal{F}_\mu^{(\leq d)}$  is defined as

$$\tilde{f}(\mathbf{X}) := f(q(\langle 0 \rangle_{\nu_q}, \mathbf{X}), \dots, q(\langle \ell_q - 1 \rangle_{\nu_q}, \mathbf{X}), w(\langle 0 \rangle_{\nu_w}, \mathbf{X}), \dots, w(\langle \ell_w - 1 \rangle_{\nu_w}, \mathbf{X})); \quad (10)$$

- the public input is consistent with the witness, that is, the public input polynomial  $p \in \mathcal{F}_\nu^{(\leq 1)}$  is identical to  $w(0^{\mu+\nu_w-\nu}, \mathbf{X}) \in \mathcal{F}_\nu^{(\leq 1)}$ .

$\mathcal{R}_{PLONK}$  is general enough to capture many computational models. In the introduction, we reviewed how  $\mathcal{R}_{PLONK}$  captures simple arithmetic circuits.  $\mathcal{R}_{PLONK}$  can be used to capture higher degree circuits with higher arity and more complex gates, including state machine computations.

*State Machines.*  $\mathcal{R}_{PLONK}$  can model state machine computations, as shown by Gabizon and Williamson [28]. A state machine execution with  $n - 1$  steps starts with an initial state  $\mathbf{state}_0 \in \mathbb{F}^k$  where  $k$  is the width of the state vector. In each step  $i \in [0, n - 1)$ , given input the previous state  $\mathbf{state}_i$  and an online input  $\mathbf{inp}_i \in \mathbb{F}$ , the state machine executes a transition function  $f$  and outputs  $\mathbf{state}_{i+1} \in \mathbb{F}^w$ . Let  $\mathcal{T} := (\mathbf{state}_0, \dots, \mathbf{state}_{n-1})$  be the execution trace and define  $\mathbf{inp}_{n-1} := \perp$ , we say that  $\mathcal{T}$  is valid for input  $(\mathbf{inp}_0, \dots, \mathbf{inp}_{n-1})$  if and only if (i)  $\mathbf{state}_{n-1}[0] = 0^k$ , and (ii)  $\mathbf{state}_{i+1} = f(\mathbf{state}_i, \mathbf{inp}_i)$  for all  $i \in [0, n - 1)$ .

We build a HyperPlonk indexed relation that captures the state machine computation. W.l.o.g we assume that  $n = 2^\mu$  for some  $\mu \in \mathbb{N}$ .<sup>7</sup> Let  $\nu_w$  be the minimal integer such that  $2^{\nu_w} > 2k$ . We also assume that there is a low-depth algebraic predicate  $f_*$  that captures the transition function  $f$ , that is,  $f_*(\mathbf{state}', \mathbf{state}, \mathbf{inp}) = 0$  if and only if  $\mathbf{state}' = f(\mathbf{state}, \mathbf{inp})$ . For each  $i \in [0, n)$ :

- the online input at the  $i$ -th step is  $\mathbf{inp}_i := w(\langle 0 \rangle_{\nu_w}, \langle i \rangle_\mu)$ ;
- the input state of step  $i$  is  $\mathbf{state}_{\mathbf{in},i} := [w(\langle 1 \rangle_{\nu_w}, \langle i \rangle_\mu), \dots, w(\langle k \rangle_{\nu_w}, \langle i \rangle_\mu)]$ ;
- the output state of step  $i$  is  $\mathbf{state}_{\mathbf{out},i} := [w(\langle k+1 \rangle_{\nu_w}, \langle i \rangle_\mu), \dots, w(\langle 2k \rangle_{\nu_w}, \langle i \rangle_\mu)]$ ;
- the selector for step  $i$  is  $\mathbf{q}_i := q(\langle i \rangle_\mu)$ ;
- the transition and output correctness are jointly captured by a high-degree algebraic map  $f'$ ,

$$f'(\mathbf{inp}_i, \mathbf{state}_{\mathbf{in},i}, \mathbf{state}_{\mathbf{out},i}; \mathbf{q}_i) := (1 - \mathbf{q}_i) \cdot f_*(\mathbf{state}_{\mathbf{out},i}, \mathbf{state}_{\mathbf{in},i}, \mathbf{inp}_i) + \mathbf{q}_i \cdot \mathbf{state}_{\mathbf{in},i}[0].$$

For all  $i \in [0, n - 1)$ , we set  $\mathbf{q}_i = 0$  so that  $\mathbf{state}_{i+1} = f_i(\mathbf{state}_i, \mathbf{inp}_i)$  if and only if

$$f'(\mathbf{inp}_i, \mathbf{state}_{\mathbf{in},i}, \mathbf{state}_{\mathbf{out},i}; \mathbf{q}_i) = f_*(\mathbf{state}_{\mathbf{out},i}, \mathbf{state}_{\mathbf{in},i}, \mathbf{inp}_i) = 0;$$

we set  $\mathbf{q}_{n-1} = 1$  so that  $\mathbf{state}_{\mathbf{in},n-1}[0] = 0$  if and only if

$$f'(\mathbf{inp}_{n-1}, \mathbf{state}_{\mathbf{in},n-1}, \mathbf{state}_{\mathbf{out},n-1}; \mathbf{q}_{n-1}) = \mathbf{state}_{\mathbf{in},n-1}[0] = 0.$$

<sup>7</sup> We can pad with dummy states if the number of steps is not a power of two.

Note that we also need to enforce equality between the  $i$ -th input state and the  $(i - 1)$ -th output state for all  $i \in [n - 1]$ . We achieve it by fixing a permutation  $\sigma$  and constraining that the witness assignment is invariant after applying the permutation.

*Remark 4.2.* We can halve the size of the witness and remove the permutation check by using the polynomial shifting technique in Sect. 3.6. Specifically, we can remove output state columns  $\text{state}_{\text{out},i}$  and replace it with  $\text{state}_{\text{in},i+1}$  for every  $i \in [0, n)$ .

### 4.2 The PolyIOP Protocol

In this Section, we present a *multivariate* PIOP for  $\mathcal{R}_{\text{PLOWK}}$  that removes expensive FFTs.

*Construction.* Intuitively, the PIOP for  $\mathcal{R}_{\text{PLOWK}}$  builds on a zero-check PIOP (Sect. 3.2) for custom algebraic gates and a permutation-check PIOP (Sect. 3.5) for copy constraints; consistency between the public input and the online witness is achieved via a random evaluation check between the public input polynomial and the witness polynomial.

Let  $\text{gp} := (\mathbb{F}, \ell, n, \ell_w, \ell_q, f)$  be the public parameters and let  $d := \deg(f)$ . For a tuple  $(\mathfrak{i}; \mathfrak{x}; \mathfrak{w}) = ((q, \sigma); (p, [[w]]); w)$ , we describe the protocol in Fig. 1.

**Indexer.**  $\mathcal{I}(q, \sigma)$  calls the permutation PIOP indexer  $([[s_{\text{id}}]], [[s_{\sigma}]]) \leftarrow \mathcal{I}_{\text{perm}}(\sigma)$ . The oracle output is  $([[q]], [[s_{\text{id}}]], [[s_{\sigma}]])$ , where  $q \in \mathcal{F}_{\mu+\nu_q}^{(\leq 1)}$ ,  $s_{\text{id}}, s_{\sigma} \in \mathcal{F}_{\mu+\nu_w}^{(\leq 1)}$ .

**The protocol.**  $\mathcal{P}(\text{gp}, \mathfrak{i}, p, w)$  and  $\mathcal{V}(\text{gp}, p, [[q]], [[s_{\text{id}}]], [[s_{\sigma}]])$  run the following protocol.

1.  $\mathcal{P}$  sends  $\mathcal{V}$  the witness oracle  $[[w]]$  where  $w \in \mathcal{F}_{\mu+\nu_w}^{(\leq 1)}$ .
2.  $\mathcal{P}$  and  $\mathcal{V}$  run a PIOP for the gate identity, which is a zero-check PIOP (Sect. 3.2) for  $([[\tilde{f}]]; \tilde{f}) \in \mathcal{R}_{\text{ZERO}}$  where  $\tilde{f} \in \mathcal{F}_{\mu}^{(\leq d)}$  is as defined in Eq. 10.
3.  $\mathcal{P}$  and  $\mathcal{V}$  run a PIOP for the wiring identity, which is a permutation PIOP (Sect. 3.5) for  $(\sigma; ([[w]], [[w]]); (w, w)) \in \mathcal{R}_{\text{PERM}}$ .
4.  $\mathcal{V}$  checks the consistency between witness and public input. It samples  $\mathbf{r} \leftarrow \mathbb{F}^{\nu}$ , queries  $[[w]]$  on input  $(\langle 0 \rangle_{\mu+\nu_w-\nu}, \mathbf{r})$ , and checks  $p(\mathbf{r}) \stackrel{?}{=} w(\langle 0 \rangle_{\mu+\nu_w-\nu}, \mathbf{r})$ .

Fig. 1. PIOP for  $\mathcal{R}_{\text{PLOWK}}$ .

**Theorem 4.3.** Let  $\text{gp} := (\mathbb{F}, \ell, n, \ell_w, \ell_q, f)$  be the public parameters where  $\ell_w, \ell_q = O(1)$  are some constants. Let  $d := \deg(f)$ . The construction in Fig. 1 is a multivariate PolyIOP for relation  $\mathcal{R}_{\text{PLOWK}}$  (Definition 4.1) with soundness error  $\mathcal{O}(\frac{2^{\mu+d\mu}}{|\mathbb{F}|})$  and the following complexity:

- the prover time is  $\text{tp}_{\text{plonk}}^{\text{gp}} = \mathcal{O}(nd \log^2 d)$ ;



- the verifier time is  $\text{tv}_{\text{plonk}}^{\text{gp}} = \mathcal{O}(\mu + \ell)$ ;
- the query complexity is  $\mathfrak{q}_{\text{plonk}}^{\text{gp}} = 2\mu + 4 + \log \ell_w$ , that is,  $2\mu + \log \ell_w$  univariate oracle queries, 3 multilinear oracle queries, and 1 query to the virtual polynomial  $\tilde{f}$ .
- the round complexity and the number of proof oracles is  $\text{rc}_{\text{plonk}}^{\text{gp}} = 2\mu + 1 + \nu_w$ ;
- the number of field elements sent by the prover is  $\text{nf}_{\text{plonk}}^{\text{gp}} = 2\mu$ ;
- the size of the proof oracles is  $\text{pl}_{\text{plonk}}^{\text{gp}} = \mathcal{O}(n)$ ; the size of the witness is  $n\ell_w$ .

*Remark 4.4.* Two separate sumcheck PIOPs are underlying the HyperPlonk PIOP. We can batch the two sumchecks into one by random linear combination. The optimized protocol has round complexity  $\mu + 1 + \log \ell_w$ , and the number of field elements sent by the prover is  $\mu$ . The query complexity  $\mu + 3 + \log \ell_w$ , that is,  $\mu + \log \ell_w$  univariate queries, 2 multilinear queries, and 1 queries to the virtual polynomial  $\tilde{f}$ .

## 5 Orion+: A Linear-Time Multilinear PCS with Constant Proof Size

Recently, Xie et al. [50] introduced a highly efficient multilinear polynomial commitment scheme called Orion. The prover time is strictly linear, that is,  $\mathcal{O}(2^\mu)$  field operations and hashes where  $\mu$  is the number of variables. For  $\mu = 27$ , it takes only 115s to commit to a polynomial and compute an evaluation proof using a single thread on a consumer-grade desktop. The verifier time and proof size is  $\mathcal{O}_\lambda(\mu^2)$ , which also improves the state-of-the-art [16,32]. However, the concrete proof size is still unsatisfactory, e.g., for  $\mu = 27$ , the proof size is 6 MBs. In this section, we describe a variant of Orion PCS that enjoys similar proving complexity but has  $\mathcal{O}(\mu)$  proof size and verifier time, with good constants. In particular, for security parameter  $\lambda = 128$  and  $\mu = 27$ , the proof size is less than 10 KBs, which is  $600\times$  smaller than Orion for  $\mu = 27$ .

In this section, we first review the linear-code-based PCS that Orion builds upon. Then we show how Orion+ shrinks the proof size and verifier time. For more details see the full version.

*Linear-Time PCS from Tensor-Product Argument* [16,32]. Bootle, Chiesa, and Groth [16] propose an elegant scheme for building PCS with strictly linear-time provers. Golovnev et al. [32] later further simplify the scheme. Let  $f \in \mathcal{F}_\mu^{(\leq 1)}$  be a multilinear polynomial where  $f_{\mathbf{b}} \in \mathbb{F}$  is the coefficient of  $\mathbf{X}_{\mathbf{b}} := \mathbf{X}_1^{b_1} \cdots \mathbf{X}_\mu^{b_\mu}$  for every  $\mathbf{b} \in B_\mu$ . Denote by  $n = 2^\mu$ ,  $k = 2^\nu < 2^\mu$  and  $m = n/k$ , one can view the evaluation of  $f$  as a tensor product, that is,

$$f(\mathbf{X}) = \langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle \quad (11)$$

where  $\mathbf{w} = (f_{\langle 0 \rangle}, \dots, f_{\langle n-1 \rangle})$ ,  $\mathbf{t}_0 = (\mathbf{X}_{\langle 0 \rangle}, \mathbf{X}_{\langle 1 \rangle}, \dots, \mathbf{X}_{\langle k-1 \rangle})$  and  $\mathbf{t}_1 = (\mathbf{X}_{\langle 0 \rangle}, \mathbf{X}_{\langle k \rangle}, \dots, \mathbf{X}_{\langle (m-1) \cdot k \rangle})$ . Here  $\langle i \rangle$  denotes the  $\mu$ -bit binary representation of  $i$ . Let  $E : \mathbb{F}^m \rightarrow \mathbb{F}^M$  be a linear encoding scheme, that is, a linear function whose image is a linear code. Golovnev et al. [32, §4.2] construct a PCS scheme as follows:

- **Commitment:** To commit a multilinear polynomial  $f$  with coefficients  $\mathbf{w} \in \mathbb{F}^n$ , the prover  $\mathcal{P}$  interprets  $\mathbf{w}$  as a  $k \times m$  matrix, namely  $\mathbf{w} \in \mathbb{F}^{k \times m}$ , encodes  $\mathbf{w}$ 's rows, and obtains matrix  $W \in \mathbb{F}^{k \times M}$  such that  $W[i, :] = E(\mathbf{w}[i, :])$  for every  $i \in [k]$ . Then  $\mathcal{P}$  computes a Merkle tree commitment for each column of  $W$  and builds another Merkle tree  $T$  on top of the column commitments. The polynomial commitment  $C_f$  is the Merkle root of  $T$ .
- **Evaluation proof:** To prove that  $f(\mathbf{z}) = y$  for some point  $\mathbf{z} \in \mathbb{F}^\mu$  and value  $y \in \mathbb{F}$ , the prover  $\mathcal{P}$  translates  $\mathbf{z}$  to vectors  $\mathbf{t}_0 \in \mathbb{F}^k$  and  $\mathbf{t}_1 \in \mathbb{F}^m$  as above and proves that  $\langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle = y$  (where  $\mathbf{w} \in \mathbb{F}^{k \times m}$  is the message encoded and committed in  $C_f$ ). To do so,  $\mathcal{P}$  does two things:
  - **Proximity check:** The prover shows that the matrix  $W \in \mathbb{F}^{k \times M}$  committed by  $C_f$  is close to  $k$  codewords. Specifically, the verifier sends a random vector  $\mathbf{r} \in \mathbb{F}^k$ , the prover replies with a vector  $\mathbf{y}_\mathbf{r} := \mathbf{r} \cdot \mathbf{w} \in \mathbb{F}^m$  which is the linear combination of  $\mathbf{w}$ 's rows according to  $\mathbf{r}$ . The verifier checks that the encoding of  $\mathbf{y}_\mathbf{r}$ , namely  $E(\mathbf{y}_\mathbf{r}) \in \mathbb{F}^M$ , is close to  $\mathbf{r} \cdot W$ , the linear combination of  $W$ 's rows. This implies that the  $k$  rows of  $W$  are all close to codewords [32, §4.2].
  - **Consistency check:** The prover shows that  $\langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle = y$  where  $\mathbf{w} \in \mathbb{F}^{k \times m}$  is the  $k$  error-decoded messages from  $W \in \mathbb{F}^{k \times M}$  committed in  $C_f$ . The scheme is similar to the proximity check except that we replace the random vector  $\mathbf{r}$  with  $\mathbf{t}_0$ . After receiving the linearly combined vector  $\mathbf{y}_0 \in \mathbb{F}^m$ , the verifier further checks that  $\langle \mathbf{y}_0, \mathbf{t}_1 \rangle = y$ .

We describe the concrete PCS evaluation protocol below.

Protocol 1 (PCS evaluation [32]): The goal is to check that  $\langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle = y$  (where  $\mathbf{w} \in \mathbb{F}^{k \times m}$  is the message encoded and committed in  $C_f$ ).

1.  $\mathcal{V}$  sends a random vector  $\mathbf{r} \in \mathbb{F}^k$ .
2.  $\mathcal{P}$  sends vector  $\mathbf{y}_\mathbf{r}, \mathbf{y}_0 \in \mathbb{F}^m$  where  $\mathbf{y}_\mathbf{r} = \sum_{i=1}^k \mathbf{r}_i \cdot \mathbf{w}[i, :]$ , and  $\mathbf{y}_0 = \sum_{i=1}^k \mathbf{t}_{0,i} \cdot \mathbf{w}[i, :]$ , where  $\mathbf{w} \in \mathbb{F}^{k \times m}$  is the message matrix being encoded and committed.
3.  $\mathcal{V}$  sends  $\mathcal{P}$  a random subset  $I \subseteq [M]$  with size  $|I| = \Theta(\lambda)$ .
4.  $\mathcal{P}$  opens the entire columns  $\{W[:, j]\}_{j \in I}$  using Merkle proofs, where  $W \in \mathbb{F}^{k \times M}$  is the row-wise encoded matrix. That is,  $\mathcal{P}$  outputs the column commitment  $h_j$  for every column  $j \in I$ , and provide the Merkle proof for  $h_j$  w.r.t. to Merkle root  $C_f$ .
5.  $\mathcal{V}$  checks that (i) the Merkle openings are correct w.r.t.  $C_f$ , and (ii) for all  $j \in I$ , it holds that  $E(\mathbf{y}_\mathbf{r})_j = \langle \mathbf{r}, W[:, j] \rangle$  and  $E(\mathbf{y}_0)_j = \langle \mathbf{t}_0, W[:, j] \rangle$ .
6.  $\mathcal{V}$  checks that  $\langle \mathbf{y}_0, \mathbf{t}_1 \rangle = y$ .

Note that by sampling a subset  $I$  with size  $\Theta(\lambda)$  and checking that  $\mathbf{r} \cdot W, \mathbf{t}_0 \cdot W$  are consistent with the encodings  $E(\mathbf{y}_\mathbf{r}), E(\mathbf{y}_0)$  on set  $I$ , the verifier is confident that  $\mathbf{r} \cdot W, \mathbf{t}_0 \cdot W$  are indeed close to the encodings  $E(\mathbf{y}_\mathbf{r}), E(\mathbf{y}_0)$  with high probability. By setting  $k = \sqrt{n}$ , the prover takes  $O(n)$   $\mathbb{F}$ -ops and hashes; the verifier time and proof size are both  $O_\lambda(\sqrt{n})$ . Orion describes an elegant code-switching scheme that reduces the proof size and verifier time down to  $O_\lambda(\log^2(n))$ . However, the concrete proof size is still large. Next, we describe a scheme that has much smaller proof.

*Linear-Time PCS with Small Proofs.* Similar to Orion (and more generally, the proof composition technique [15, 16, 32]), instead of letting the verifier check the correctness of  $\mathbf{y}_r, \mathbf{y}_0$  and the openings of the columns  $W[:, j] \forall j \in I$ , the prover can compute another (succinct) outer proof validating the correctness of  $\mathbf{y}_r, \mathbf{y}_0, W[:, j]$ . However, we need to minimize the outer proof’s circuit complexity, which is non-trivial. Orion builds an efficient SNARK circuit that removes all of the hashing gadgets, with the tradeoff of larger proof size. We describe a variant of their scheme that minimizes the proof size without significantly increasing the circuit complexity.

Specifically, after receiving challenge vector  $\mathbf{r} \in \mathbb{F}^k$ ,  $\mathcal{P}$  instead sends  $\mathcal{V}$  commitments  $C_r, C_0$  to the messages  $\mathbf{y}_r, \mathbf{y}_0$ ; after receiving  $\mathcal{V}$ ’s random subset  $I \subset [M]$ ,  $\mathcal{P}$  computes a SNARK proof for the following statement:

Statement 1 (PCS Eval verification):

- Witness:  $\mathbf{y}_r, \mathbf{y}_0 \in \mathbb{F}^m, \{W[:, j]\}_{j \in I}$ .
- Circuit statements:
  - $C_r, C_0$  are the commitments to  $\mathbf{y}_r, \mathbf{y}_0$  respectively.
  - For all  $j \in I$ , it holds that
    - \*  $h_j = H(W[:, j])$  where  $H$  is a fast hashing scheme;
    - \*  $E(\mathbf{y}_r)_j = \langle \mathbf{r}, W[:, j] \rangle$  and  $E(\mathbf{y}_0)_j = \langle \mathbf{t}_0, W[:, j] \rangle$ .
  - $\langle \mathbf{y}_0, \mathbf{t}_1 \rangle = y$ .
- Public output:  $\{h_j\}_{j \in I}$ , and  $C_r, C_0$ .

Besides the SNARK proof, the prover also provides the openings of  $\{h_j\}_{j \in I}$  with respect to the commitments  $C_f$ . Intuitively, the new protocol is “equivalent” to Protocol 1, because the SNARK witness  $\{W[:, j]\}_{j \in I}$  and  $\mathbf{y}_r, \mathbf{y}_0$  are identical to those committed in  $C_f, C_r, C_0$  by the binding property of the commitments; and the SNARK does all of the verifier checks. Unfortunately, the scheme has the following drawbacks:

- Instantiating the commitments with Merkle trees leads to a large overhead on the proof size. In particular, the proof contains  $|I|$  Merkle proofs, each with length  $O(\log n)$ . For 128-bit security, we need to set  $|I| = 1568$ , and the proof size is at least 1 MBs for  $\mu = 20$ .
- The random subset  $I$  varies for different evaluation instances. It is non-trivial to efficiently lookup the witness  $\{E(\mathbf{y}_r)_j, E(\mathbf{y}_0)_j\}_{j \in I}$  in the circuit if the set  $I$  is dynamic (i.e. we need an efficient random access gadget).
- The circuit complexity is huge. In particular, the circuit is dominated by the commitments to  $\mathbf{y}_r, \mathbf{y}_0$  and the hash commitments to  $\{W[:, j]\}_{j \in I}$ . This leads to  $2m + k|I|$  hash gadgets in the circuit. Note that we can’t use algebraic hash functions like Rescue [1] or Poseidon [33], which are circuit-friendly, but have slow running times. For  $\mu = 26, k = m = \sqrt{n}$  and 128-bit security (where  $|I| = 1568$ ), this leads to 13 million hash gadgets where each hash takes hundreds to thousands of constraints, which is unaffordable.

We resolve the above issues via the following observations.

First, a large portion of the multilinear PCS evaluation proof is Merkle opening paths. We can shrink the proof size by replacing Merkle trees with multilinear

PCS that enable efficient batch openings (Sect. 3.7). Specifically, in the committing phase, after computing the hashes of  $W$ 's columns, instead of building another Merkle tree  $T$  of size  $M = O(n/k)$  and set the Merkle root as the commitment, the prover can commit to the column hashes using a multilinear PCS (e.g. KZG). Though the KZG committing is more expensive, *the problem size has been reduced to  $O(n/k)$ , thus for sufficiently large  $k$ , the committing complexity is still approximately  $O(n)$   $\mathbb{F}$ -ops.* A great advantage is that the batch opening proof for  $\{h_j\}_{j \in I}$  consists of only  $O(\log n)$  group/field elements, with good constant. Even better, when instantiating the outer proof with HyperPlonk(+), the openings can be batched with those in the outer SNARK and thus incur almost no extra cost in proof size.

Second, with Plookup, we can efficiently simulate random access in arrays in the SNARK circuit. For example, to extract witness  $\{\mathbf{Y}_{\mathbf{r},j} = E(\mathbf{y}_{\mathbf{r}})_j\}_{j \in I}$ , we can build an (online) table  $T$  where each element of the table is a pair  $(i, E(\mathbf{y}_{\mathbf{r}})_i)$  ( $1 \leq i \leq M$ ). Then for every  $j \in I$ , we build a lookup gate checking that  $(j, \mathbf{Y}_{\mathbf{r},j})$  is in the table  $T$ , thus guarantee that  $\mathbf{Y}_{\mathbf{r},j}$  is identical to  $E(\mathbf{y}_{\mathbf{r}})_j$ . The circuit description is now independent of the random set  $I$  and we only need to preprocess the circuit once in the setup phase.

Third, with the help of Commit-and-Prove-SNARKs (CP-SNARK) [2, 20, 21], there is no need to check the consistency between commitments  $C_{\mathbf{r}}, C_0$  and  $\mathbf{y}_{\mathbf{r}}, \mathbf{y}_0$  in the circuit. Instead, we can commit  $(\mathbf{y}_{\mathbf{r}}, \mathbf{y}_0)$  to a multilinear commitment  $C$ , and build a CP-SNARK proof showing that the vector underlying  $C$  is identical to the witness vector  $(\mathbf{y}_{\mathbf{r}}, \mathbf{y}_0)$  in the circuit. We further observe that  $C$  can be a part of the witness polynomials, which further removes the need of an additional CP-SNARK proof.

After applying previous optimizations, the proof size is dominated by the  $|I|$  field elements  $\{h_j\}_{j \in I}$ . We can altogether remove them by applying the CP-SNARK trick again. In particular, since  $\{h_j\}_{j \in I}$  are both committed in the polynomial commitment  $C_f$  and the SNARK witness commitment, it is sufficient to construct a CP-SNARK proving that they are consistent in the two commitments with respect to set  $I$ . We refer to the full version for constructing CP-SNARK proofs from multilinear commitments.

Since the bulk of verification work is delegated to the prover, there is no need to set  $k = \sqrt{n}$ . Instead, we can set an appropriate  $k = \Theta(\lambda/\log n)$  to minimize the outer circuit size. In particular, the circuit is dominated by 2 linear encodings (of length  $n/k$ ) and  $|I|$  hashes (of length  $k$ ). If we use vanilla HyperPlonk+ as the outer SNARK scheme and use Reinforced Concrete [5] as the hashing scheme that has a similar running time to SHA-256, for  $\mu = 30$ ,  $k = 64$  and 128-bit security (where  $|I| = 1568$ ), the circuit complexity is only  $\approx 2^{26}$  constraints. And we can expect the running time of the outer proof to be  $O_{\lambda}(n)$ .

## References

1. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szeponiec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symm. Cryptol.* **2020**(3), 1–45 (2020). <https://doi.org/10.13154/tosc.v2020.i3.1-45>
2. Aranha, D.F., Benedsen, E.M., Campanelli, M., Ganesh, C., Orlandi, C., Takahashi, A.: ECLIPSE: enhanced compiling method for pedersen-committed zkSNARK engines. *Cryptology ePrint Archive*, Report 2021/934 (2021). <https://eprint.iacr.org/2021/934>
3. Arun, A., Ganesh, C., Lokam, S., Mopuri, T., Sridhar, S.: Dew: transparent constant-sized zkSNARKs. *Cryptology ePrint Archive*, Report 2022/419 (2022). <https://eprint.iacr.org/2022/419>
4. Babai, L., Moran, S.: Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.* **36**(2), 254–276 (1988)
5. Barbara, M., et al.: Reinforced concrete: fast hash function for zero knowledge proofs and verifiable computation. *Cryptology ePrint Archive*, Report 2021/1038 (2021). <https://eprint.iacr.org/2021/1038>
6. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) *EUROCRYPT 2012*. LNCS, vol. 7237, pp. 263–280. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-29011-4\\_17](https://doi.org/10.1007/978-3-642-29011-4_17)
7. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) *ICALP 2018*. LIPIcs, vol. 107, pp. 14:1–14:17. Schloss Dagstuhl, July 2018. <https://doi.org/10.4230/LIPIcs.ICALP.2018.14>
8. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) *CRYPTO 2019*. LNCS, vol. 11694, pp. 701–732. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-26954-8\\_23](https://doi.org/10.1007/978-3-030-26954-8_23)
9. Ben-Sasson, E., Carmon, D., Kopparty, S., Levit, D.: Elliptic curve fast fourier transform (ECFFT) part ii: scalable and transparent proofs over all large fields (2022)
10. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) *EUROCRYPT 2019*. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17653-2\\_4](https://doi.org/10.1007/978-3-030-17653-2_4)
11. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) *TCC 2016*. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53644-5\\_2](https://doi.org/10.1007/978-3-662-53644-5_2)
12. Ben-Sasson, E., Sudan, M.: Short pcps with polylog query complexity. *SIAM J. Comput.* **38**(2), 551–607 (2008)
13. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Goldwasser, S. (ed.) *ITCS 2012*, pp. 326–349. ACM, January 2012. <https://doi.org/10.1145/2090236.2090263>
14. Bitansky, N., Chiesa, A., Ishai, Y., Paneth, O., Ostrovsky, R.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) *TCC 2013*. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-36594-2\\_18](https://doi.org/10.1007/978-3-642-36594-2_18)

15. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10626, pp. 336–365. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-70700-6\\_12](https://doi.org/10.1007/978-3-319-70700-6_12)
16. Bootle, J., Chiesa, A., Groth, J.: Linear-time arguments with sublinear verification from tensor codes. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12551, pp. 19–46. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-64378-2\\_2](https://doi.org/10.1007/978-3-030-64378-2_2)
17. Bootle, J., Chiesa, A., Hu, Y., Orrù, M.: Gemini: elastic SNARKs for diverse environments. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 427–457. Springer, Heidelberg, May/June 2022. [https://doi.org/10.1007/978-3-031-07085-3\\_15](https://doi.org/10.1007/978-3-031-07085-3_15)
18. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press, May 2018. <https://doi.org/10.1109/SP.2018.00020>
19. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 677–706. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45721-1\\_24](https://doi.org/10.1007/978-3-030-45721-1_24)
20. Campanelli, M., Faonio, A., Fiore, D., Querol, A., Rodríguez, H.: Lunar: a toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13092, pp. 3–33. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-92078-4\\_1](https://doi.org/10.1007/978-3-030-92078-4_1)
21. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: modular design and composition of succinct zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2075–2092. ACM Press, November 2019. <https://doi.org/10.1145/3319535.3339820>
22. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: HyperPlonk: plonk with linear-time prover and high-degree custom gates. Cryptology ePrint Archive, Report 2022/1355 (2022). <https://eprint.iacr.org/2022/1355>
23. Chiesa, A., Forbes, M.A., Spooner, N.: A zero knowledge sumcheck and its applications. Cryptology ePrint Archive, Report 2017/305 (2017). <https://eprint.iacr.org/2017/305>
24. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 738–768. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45721-1\\_26](https://doi.org/10.1007/978-3-030-45721-1_26)
25. Drake, J.: Plonk-style SNARKs without FFTs (2019). <https://notes.ethereum.org/DLRqK9V7RIOsTZkab8HmQ?view>
26. Gabizon, A.: Multiset checks in plonk and pllookup. <https://hackmd.io/@arielg/ByFgSDA7D>
27. Gabizon, A., Williamson, Z.J.: pllookup: a simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315 (2020). <https://eprint.iacr.org/2020/315>
28. Gabizon, A., Williamson, Z.J.: Proposal: the turbo-plonk program syntax for specifying snark programs (2020). <https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo-plonk.pdf>
29. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019). <https://eprint.iacr.org/2019/953>

30. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-38348-9\\_37](https://doi.org/10.1007/978-3-642-38348-9_37)
31. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18**(1), 186–208 (1989)
32. Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R.S.: Brakedown: linear-time and post-quantum SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/1043 (2021). <https://eprint.iacr.org/2021/1043>
33. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: a new hash function for zero-knowledge proof systems. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021, pp. 519–535. USENIX Association, August 2021
34. 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](https://doi.org/10.1007/978-3-662-49896-5_11)
35. Harvey, D., Van Der Hoeven, J.: Polynomial multiplication over finite fields in time. *J. ACM (JACM)* **69**(2), 1–40 (2022)
36. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-17373-8\\_11](https://doi.org/10.1007/978-3-642-17373-8_11)
37. Kattis, A.A., Panarin, K., Vlasov, A.: RedShift: transparent SNARKs from list polynomial commitments. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022, pp. 1725–1737. ACM Press, November 2022. <https://doi.org/10.1145/3548606.3560657>
38. Lee, J.: Dory: efficient, transparent arguments for generalised inner products and polynomial commitments. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13043, pp. 1–34. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-90453-1\\_1](https://doi.org/10.1007/978-3-030-90453-1_1)
39. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. *J. ACM (JACM)* **39**(4), 859–868 (1992)
40. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 222–242. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-36594-2\\_13](https://doi.org/10.1007/978-3-642-36594-2_13)
41. Pearson, L., Fitzgerald, J., Masip, H., Bellés-Muñoz, M., Muñoz-Tapia, J.L.: PlonKup: reconciling PlonK with plookup. Cryptology ePrint Archive, Report 2022/086 (2022). <https://eprint.iacr.org/2022/086>
42. Posen, J., Kattis, A.A.: Caulk+: table-independent lookup arguments. Cryptology ePrint Archive, Report 2022/957 (2022). <https://eprint.iacr.org/2022/957>
43. Setty, S.: Spartan: efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 704–737. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-56877-1\\_25](https://doi.org/10.1007/978-3-030-56877-1_25)
44. Setty, S., Lee, J.: Quarks: quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275 (2020). <https://eprint.iacr.org/2020/1275>
45. System, E.: Jellyfish jellyfish cryptographic library (2022). <https://github.com/EspressoSystems/jellyfish>
46. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 71–89. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40084-1\\_5](https://doi.org/10.1007/978-3-642-40084-1_5)
47. Thaler, J.: Proofs, arguments, and zero-knowledge (2020)

48. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy, pp. 926–943. IEEE Computer Society Press, May 2018. <https://doi.org/10.1109/SP.2018.00060>
49. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 733–764. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-26954-8\\_24](https://doi.org/10.1007/978-3-030-26954-8_24)
50. Xie, T., Zhang, Y., Song, D.: Orion: zero knowledge proof with linear prover time. Cryptology ePrint Archive, Report 2022/1010 (2022). <https://eprint.iacr.org/2022/1010>
51. Xie, T., Zhang, Y., Song, D.: Orion: zero knowledge proof with linear prover time. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 299–328. Springer, Heidelberg, August 2022. [https://doi.org/10.1007/978-3-031-15985-5\\_11](https://doi.org/10.1007/978-3-031-15985-5_11)
52. Xiong, A.L., et al.: VERI-ZEXE: decentralized private computation with universal setup. Cryptology ePrint Archive, Report 2022/802 (2022). <https://eprint.iacr.org/2022/802>
53. Zapico, A., Buterin, V., Khovratovich, D., Maller, M., Nitulescu, A., Simkin, M.: Caulk: lookup arguments in sublinear time. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022, pp. 3121–3134. ACM Press, November 2022. <https://doi.org/10.1145/3548606.3560646>
54. Zcash: PLONKish arithmetization. <https://zcash.github.io/halo2/concepts/arithmetization.html> (2022)
55. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: 2020 IEEE Symposium on Security and Privacy, pp. 859–876. IEEE Computer Society Press, May 2020. <https://doi.org/10.1109/SP40000.2020.00052>





# Spartan and Bulletproofs are Simulation-Extractable (for Free!)

Quang Dao<sup>1</sup>(✉) and Paul Grubbs<sup>2</sup>

<sup>1</sup> Carnegie Mellon University, Pittsburgh, USA  
qvd@andrew.cmu.edu

<sup>2</sup> University Of Michigan, Ann Arbor, USA  
paulgrub@umich.edu

**Abstract.** Increasing deployment of advanced zero-knowledge proof systems, especially zkSNARKs, has raised critical questions about their security against real-world attacks. Two classes of attacks of concern in practice are *adaptive soundness* attacks, where an attacker can prove false statements by choosing its public input after generating a proof, and *malleability* attacks, where an attacker can use a valid proof to create another valid proof it could not have created itself. Prior work has shown that *simulation-extractability* (SIM-EXT), a strong notion of security for proof systems, rules out these attacks.

In this paper, we prove that two transparent, discrete-log-based zkSNARKs, Spartan and Bulletproofs, are simulation-extractable (SIM-EXT) in the random oracle model if the discrete logarithm assumption holds in the underlying group. Since these assumptions are required to prove standard security properties for Spartan and Bulletproofs, our results show that SIM-EXT is, surprisingly, “for free” with these schemes. Our result is the first SIM-EXT proof for Spartan and encompasses both linear- and sublinear-verifier variants. Our result for Bulletproofs encompasses both the aggregate range proof and arithmetic circuit variants, and is the first to not rely on the algebraic group model (AGM), resolving an open question posed by Ganesh et al. (EUROCRYPT’22). As part of our analysis, we develop a generalization of the tree-builder extraction theorem of Attema et al. (TCC’22), which may be of independent interest.

## 1 Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) allow a computationally-bounded prover to produce a proof about a NP statement *without* revealing anything other than its validity, and with proof size *sublinear* in the size of the witness [12, 32, 34]. An important line of recent works [7, 9, 13, 15, 17, 20, 32, 35, 38, 44, 48, 57, 60] has produced concretely efficient constructions of zkSNARKs for range proofs (e.g., Bulletproofs [16]) and general arithmetic circuit satisfiability (e.g., Spartan [54]) that have seen widespread deployment, especially in blockchains and cryptocurrencies [1, 8, 21, 51, 53, 56, 61], along with potential deployment in other areas of interests [40].

---

Q. Dao—Part of the work was done while the first author was at the University of Michigan.

As zkSNARKs are deployed in practice, it is important to understand whether they are actually secure against the kinds of attacks they are likely to face in real systems. Two security properties in particular give us pause: first, *adaptive soundness*, where a malicious prover must be unable to prove false statements even if it chooses the input after generating a proof; a related notion, adaptive knowledge soundness, guarantees extraction is possible against such an adaptive prover. The second property is *non-malleability*, where an accepting proof cannot be modified into a different one without knowing the witness. Neither property is implied by standard security definitions like non-adaptive (knowledge) soundness and zero knowledge, and schemes lacking these properties have been attacked in practice. For example, the voting system Helios was broken by an adaptive soundness attack on a zero-knowledge proof [11]; subsequent work found similar issues with the SwissPost voting system [41] for government elections. Though not against zero-knowledge proofs directly, malleability attacks are common in cryptocurrencies: for example, a malleability attack was allegedly used<sup>1</sup> to steal hundreds of millions of dollars from MtGox [47].

Fortunately, a security property called *simulation extractability* (SIM-EXT) implies adaptive (knowledge) soundness and non-malleability for zkSNARKs. Intuitively, SIM-EXT requires that the knowledge extractor succeeds even when the malicious prover can request *simulated* proofs for *arbitrary* statements. If we could prove zkSNARKs that are already used (or are likely to be used) in practice are SIM-EXT, we could be more confident they would resist advanced attacks that use adaptivity or malleability. Ideally, we could prove SIM-EXT in idealized models (e.g., the random oracle model, or ROM) and using assumptions (e.g. discrete-log), which are sufficient to prove standard security guarantees for zkSNARKs; this would indicate SIM-EXT comes (roughly) “for free”.

A pair [29,30] of beautiful recent works by Ganesh et al. on SIM-EXT for zkSNARKs lays a path towards this goal. In [29], the authors give a general SIM-EXT theorem for zkSNARKs with updatable SRS, and use it to show PlonK [28], Marlin [20], and Sonic [45] are all SIM-EXT. In [30], the authors show SIM-EXT for Bulletproofs. Unfortunately, these works do not get us all the way towards our goal: first, because their techniques do not extend to *transparent* zkSNARKs like Spartan, which use different building blocks; second, because their results rely on the algebraic group model (AGM) [27] and are not currently known to hold from discrete log in the ROM.

## 1.1 Our Results

In this paper we prove that Spartan and Bulletproofs, two state-of-the-art transparent zkSNARKs, satisfy SIM-EXT in the ROM assuming only that the discrete log assumption holds. Our analyses required developing some new technical tools which may be of independent interest. Since Spartan and Bulletproofs were originally analyzed in the ROM and rely on the discrete log assumption,

---

<sup>1</sup> A later study [22] cast some doubt on these claims, but did find evidence that over three hundred thousand Bitcoins had been involved in malleability attacks.

our results imply these protocols are SIM-EXT “for free”—unmodified and without additional assumptions or stronger idealized models. More precisely, we prove SIM-EXT for two variants of Spartan—Spartan-NIZK, which has linear verifier time, and Spartan-SNARK, which has sublinear verifier time—instantiated with the default Hyrax-based polynomial commitment scheme [57]. These are the first proofs of SIM-EXT for any Spartan variant; we believe the Spartan-SNARK result is also the first proof of SIM-EXT for *any* transparent zkSNARK with sublinear verifier time. Similarly, we prove SIM-EXT for two versions of Bulletproofs—the aggregate range proof protocol BP-ARP used in several cryptocurrencies [36, 46] and the arithmetic circuit satisfiability proof BP-ACSPf. Our proofs for these protocols are the first that do not rely on the algebraic group model.

Our results help to build confidence that state-of-the-art and deployed zkSNARKs resist the kinds of attacks these protocols will face as they see wider deployment in the future. Of more theoretical interest, they also imply the surprising fact that, in the ROM, a powerful primitive like a SIM-EXT zkSNARK can be built from a very weak assumption like discrete log.

The proofs of these four theorems are nontrivial; to prove them we built several new technical tools that may be of independent interest for future SIM-EXT analyses. We extended prior security notions for SIM-EXT to the transparent NIZK setting. We also needed to develop a nontrivial generalization of the tree extractor of Attema et al. [2].

Our analyses are also done with an emphasis on concrete security. Where possible we try to explicitly measure adversarial runtime and success probability. We also evaluate our bounds to estimate bit security for typical parameters for Spartan and Bulletproofs, and compare the bit security we obtain against other analyses where possible. Our bounds inherit the non-tightness common to most rewinding-based knowledge soundness analyses of NIZKs, and so the provable SIM-EXT security we get (in terms of bits) is quite low. Nevertheless, we believe our results can be improved by future work, and hope they eventually inform future parameter selection processes for zkSNARK standards [62].

## 1.2 Technical Overview

We follow the high-level approach to proving SIM-EXT developed by [24] and further generalized in [29, 30]: for a Fiat-Shamir-compiled argument  $\Pi_{\text{FS}}$ , SIM-EXT is implied by three other properties: (1) adaptive knowledge soundness, (2) a form of zero knowledge, and (3) a unique-response property. Since the results in [24] are specific to  $\Sigma$ -protocols and those in [30] are specific to the AGM, we take the SIM-EXT theorem of [29] as our starting point. After suitable adaptations to the transparent setting—we give these in Sect. 3—this theorem says that  $\Pi_{\text{FS}}$  is SIM-EXT in the ROM if:

1. it is adaptively knowledge sound (hereafter we will omit “adaptive” if it is clear from context),
2. it is perfect  $k$ -ZK, meaning that there exists a simulator that perfectly simulates honest proofs, but only programs the RO when generating the  $k$ -th challenge,

3. it is  $k$ -UR for the *same* round  $k$ , meaning no adversary can produce two accepting proofs that are identical up to the  $k$ -th round, *even if* it can program that round's challenge.

Proving these three properties is challenging, and required us to develop novel techniques which we summarize below.

*Knowledge Soundness.* We prove knowledge soundness for non-interactive versions of Spartan and Bulletproofs using a standard chain of reductions: namely, we reduce to the *special soundness* of the underlying interactive argument. Intuitively, special soundness of a proof system refers to the ability of an extractor to extract a witness from a *tree* of accepting transcripts with suitable structure. For multi-round protocols, special soundness is parameterized by a vector  $(n_1, n_2, \dots, n_r)$  describing the needed structure: each node at level one must have  $n_1$  outgoing edges, level two nodes have  $n_2$  edges, etc. Recently, Attema et al. [2] proved that knowledge soundness of the Fiat-Shamir-compiled argument  $\Pi_{FS}$  follows from special soundness of  $\Pi$ . We take it as our starting point; unfortunately, we cannot apply it directly to either Spartan or Bulletproofs. There are two main reasons for this: first, Attema et al. only consider perfect special soundness, but both Spartan and Bulletproofs only satisfy *computational* special soundness—roughly, because an extractor could fail to extract a witness from a tree of transcripts if a malicious prover finds a nontrivial discrete log relation.

The second reason is more subtle, and has to do with ensuring the tree has the right structure for extraction to be possible. In Attema et al., each node of the transcript tree is a prover message whose outgoing edges are labeled with distinct verifier challenges. For certain rounds in both Spartan and Bulletproofs, these verifier challenges must satisfy an extra predicate (beyond distinctness) for extraction to be possible. The tree-builder by Attema et al. does not support outputting such trees with extra structure.

To address these limitations, in Sect. 4 we give a generalization of Attema et al.'s tree-builder that has the desired properties. Our generalization captures other predicates on verifier challenges using the notion of an *efficiently-decidable partition* of the space of challenges. Intuitively, we build a wrapper algorithm that sits between the prover and the Attema et al. tree-builder, and ensures the tree has the right structure by enforcing a partition of the challenge space.

Armed with this generalization, we prove computational special soundness for all variants of Spartan and Bulletproofs, which in turn implies knowledge soundness for their Fiat-Shamir-compiled versions. In both cases, our generalized tree-builder is a crucial component: for example, special soundness of Bulletproofs requires verifier challenges to be distinct modulo  $\pm 1$ , and Spartan requires linear independence for batching challenges sent during the sumcheck subprotocol.

*Building  $k$ -ZK Simulators.* For SIM-EXT, we must prove that Spartan and Bulletproofs are perfect  $k$ -ZK, meaning their proofs can be simulated by a simulator that can only program the RO in a single round. This is a departure from the typical way to build NIZK simulators, which typically reprogram the RO

in every round; in particular, doing this for Spartan and Bulletproofs requires giving entirely new simulators for these constructions.

We build our  $k$ -ZK simulator for Bulletproofs using an approach similar to [29]. Our  $k$ -ZK simulator construction for Spartan-NIZK uses a novel strategy that is worth highlighting here: it delays the round at which the RO is reprogrammed as late as possible in the protocol (in fact, our simulator only needs to reprogram the very last verifier challenge). Another interesting aspect of our  $k$ -ZK simulator for Spartan is that the same simulator works for both Spartan-NIZK and Spartan-SNARK—though the two protocols have major differences, we observe that the parts of Spartan-SNARK that work differently than Spartan-NIZK consist entirely of evaluating (extensions of) public matrices at a public point, and so are trivially simulatable.

*$k$ -Unique Response.* To finish, we need to show Spartan and Bulletproofs are  $k$ -UR for the same  $k$  as their respective  $k$ -ZK simulators. For Spartan variants, this is straightforward—we need only reprogram the RO during the final  $\Sigma$ -subprotocol, and it is well known [24] that  $\Sigma$ -protocols satisfy unique response.

For BP-ARP and BP-ACSPf, proving  $k$ -UR is more challenging. Indeed, prior work relied heavily on the AGM for analyzing unique response—for example, [30] observe that proving their version of unique response is the only part of their analysis that seems to actually rely on the AGM, and [29] need the AGM to show that KZG polynomial commitments are unique response.

We prove  $k$ -UR for Bulletproofs using a new proof strategy that, intuitively, replaces the AGM with extraction. In more detail, we extract witnesses from both proofs output by the  $k$ -UR adversary, then argue that *either* the witnesses are the same *or* the adversary has found a discrete log relation. To finish, we use the (novel) result that the Bulletproofs inner-product argument has unique proofs. Thus, if the witnesses are the same, the proofs must be the same as well.

*Limitations and open questions.* Our results do have some important limitations. Notably, our emphasis on removing the AGM means that the tightness of our Bulletproofs results is worse than the comparable result of [30]. While this is inherent in some sense because our extractors use rewinding instead of straight-line extraction, it means that the bit security of Bulletproofs and Spartan we could prove with typical parameters would come out to be quite poor. We discuss this in Sect. 7.

An interesting open problem we leave to future work is generalizing our techniques to other transparent zkSNARKs. In particular, there is a great deal of commonality between our proofs for Spartan and Bulletproofs which could be abstracted out and proven more generally. As many later works [35, 44, 55, 60] have built on Spartan viewed as a *polynomial IOP* [17, 20], it would be interesting to generalize our analyses into a SIM-EXT framework for polynomial IOPs.

### 1.3 Related Work

Simulation-extractability (SIM-EXT) for NIZKs was first defined in [52] (using different terminology). Thereafter, a long line of work refined and studied SIM-EXT [24, 49], built SIM-EXT NIZKs [37], and showed that SIM-EXT is sufficient for other primitives like signatures of knowledge [19]. Other concurrent works attacked security of NIZKs in deployed systems, such as the voting system Helios, showing the importance of adaptive soundness [11] which is implied by SIM-EXT. Other work has looked at UC security for NIZKs [18] and given results on SIM-EXT in the QROM [23]. These works are not relevant to our results, since SIM-EXT does not imply UC security in the ROM; further, we study zkSNARKs built from discrete log, which is broken by quantum attacks.

The simulation-extractability of zkSNARKs is comparatively less well-studied. Two important prior works [29, 30] which rely on the algebraic group model [27] (AGM) are described above; [30] proves SIM-EXT of Bulletproofs, and [29] proves SIM-EXT of Plonk [28], Marlin [20], and Sonic [45].

Other work has investigated generic transforms for achieving SIM-EXT from any zkSNARK [5], particularly focused on SIM-EXT transforms for the Groth16 zkSNARK [3, 4]. Since Groth16 [38] is built using a different approach than either Spartan or Bulletproofs, and relies on non-falsifiable knowledge assumptions or the AGM, our results are incomparable to theirs.

Our paper analyzes SIM-EXT for Bulletproofs [16] and Spartan [54], two transparent zkSNARKs built from discrete-log assumptions. There is a line of related work building similar SNARKs, such as Hyrax [57], and extensions to recursive composition like Halo [15] and Nova [44]. We suspect our techniques would extend to these constructions, and leave extending them to future work.

A key technical tool our results rely on is a “tree-builder” for proving knowledge soundness of NIZKs built from multi-round interactive arguments. As described above, our approach is a generalization of a beautiful recent work by Attema et al. [2]. This work develops a tree-builder for *perfect* special sound protocols which are extractable given a tree of *distinct* verifier challenges; we generalize their result to support *computational* special soundness and to allow different conditions on verifier challenges. Wikstrom [59] gives an alternate construction and analysis of a tree-builder which could have served as a starting point for us; however, their extractor has a worse concrete running time and tightness than Attema et al. In a revision of [17], the authors generalize Attema et al.’s tree builder to handle general predicates on prover messages; since we need more general predicates on verifier challenges, their generalization is not directly useful to us. Other recent works [33, 42] analyze the knowledge soundness of Bulletproofs in the AGM/GGM without using an explicit tree-builder by, for example, going through the notion of round-by-round soundness [10].

*Concurrent work.* After the acceptance of this paper, Ganesh et al. [31] updated their ePrint version to contain a proof that Bulletproofs satisfy SIM-EXT in the ROM, removing the need for the AGM as in their conference version [30]. We note that their technique is somewhat different from ours, and that our results

additionally include proving that Spartan satisfies SIM-EXT. We leave a more detailed comparison of our work with theirs to the full version.

## 2 Preliminaries

We use  $\mathbb{F}$  to denote a finite field with  $\mathbb{F}^* = \mathbb{F} - \{0\}$ , and  $\lambda$  to denote the security parameter. For  $k, n \in \mathbb{N}$ , we denote  $[k, n] = \{k, k + 1, \dots, n\}$ , and  $[n] = [1, n]$ . We denote uniform sampling from a set  $S$  by  $a \stackrel{\$}{\leftarrow} S$ . We denote vectors by boldface, e.g.  $\mathbf{g} = (g_1, \dots, g_n)$ , and write  $\mathbf{g}^{\mathbf{a}}$  to mean  $g_1^{a_1} \dots g_n^{a_n}$ . We denote the length of a vector  $\mathbf{a}$  by  $|\mathbf{a}|$ , the inner product between two vectors  $\mathbf{a}, \mathbf{b}$  by  $\mathbf{a} \cdot \mathbf{b}$  or  $\langle \mathbf{a}, \mathbf{b} \rangle$ , the Hadamard (entry-wise) product by  $\mathbf{a} \circ \mathbf{b}$ , and the tensor product by  $\mathbf{a} \otimes \mathbf{b} = (a_1 b_1, \dots, a_1 b_m, \dots, a_n b_1, \dots, a_n b_m)$ .

Our relations are of the form  $\mathcal{R} \subseteq \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^*$  and are efficiently decidable, e.g. there exists a deterministic polynomial time algorithm that given  $(\mathbf{pp}, x, w)$  outputs whether  $(\mathbf{pp}, x, w) \in \mathcal{R}$ . We abbreviate PPT for probabilistic polynomial time, and EPT for expected (probabilistic) polynomial time.

We use *code-based games* [6] to define many of our security notions. A game  $\mathcal{G}_S^{A_1, \dots, A_n}$  denotes a run of parties  $\mathcal{A}_1, \dots, \mathcal{A}_n$  on a pre-specified set of procedures given by  $S$ , returning a bit  $b \in \{0, 1\}$ . We denote  $\Pr[\mathcal{G}_S^{A_1, \dots, A_n}]$  the probability over the random coins used by  $S$  and all adversaries that the game’s output is 1.

### 2.1 Assumptions

We assume the existence of a group generator generating *global public parameters*  $\mathbf{pp}_G := (\mathbb{G}, \mathbb{F}) \leftarrow \text{GroupGen}(1^\lambda)$ , where  $\mathbb{G}$  is a group of prime order, with  $\mathbb{F}$  as the corresponding field. These global parameters are used in the setup phase of every protocol we consider. We also assume a generator sampling procedure  $g_1, \dots, g_n \stackrel{\$}{\leftarrow} \text{GenSamp}(\mathbb{G}, n)$ . For space reasons, we omit definitions of the standard discrete log (DL) and DL relation assumptions, and refer to reader to [33].

### 2.2 Interactive Arguments

We define an interactive argument for relation  $\mathcal{R} \subseteq \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^*$ .

**Definition 2.1.** *An interactive argument for a relation  $\mathcal{R}$  is a tuple of PPT algorithms  $\Pi = (\text{Setup}, \mathcal{P}, \mathcal{V})$  with the following syntax:*

- $\text{Setup}(\mathbf{pp}_G) \rightarrow \mathbf{pp}$  : *outputs public parameters  $\mathbf{pp}$  given global parameters  $\mathbf{pp}_G$ ,*
- $\langle \mathcal{P}(w), \mathcal{V} \rangle(\mathbf{pp}, x) \rightarrow \{0, 1\}$  : *an interactive protocol whereby the prover  $\mathcal{P}$ , holding a witness  $w$ , interacts with the verifier  $\mathcal{V}$  on common input  $(\mathbf{pp}, x)$  to convince  $\mathcal{V}$  that  $(\mathbf{pp}, x, w) \in \mathcal{R}$ . At the end,  $\mathcal{V}$  outputs a bit for accept/reject.*

In the definition above, we assume the existence of a global setup algorithm  $\mathbf{pp}_G \leftarrow \text{GlobalSetup}(1^\lambda)$  (see Sect. 2.1), run once and for all before the setup phase of any interactive argument. For space reasons, we refer to reader to e.g. [17] for standard definitions of completeness, knowledge soundness, and honest-verifier zero knowledge for interactive arguments.

**Definition 2.2 (Public-Coin).** *An interactive argument  $\Pi = (\text{Setup}, \mathcal{P}, \mathcal{V})$  is public-coin if in each round  $i$  the verifier  $\mathcal{V}$  samples its message uniformly at random from some challenge space  $\text{Ch}_i$ , and uses no other randomness.*

Any public-coin interactive argument has a general  $(2r+2)$ -message, or equivalently,  $(r + 1)$ -round format where the verifier sends the 0-th message, and the prover sends the last message. In particular, the transcript is of the form  $\text{tr} = (c_0, a_1, c_1, \dots, a_r, c_r, a_{r+1})$ , where  $(a_1, \dots, a_{r+1})$  are the prover’s messages and  $(c_0, \dots, c_r)$  are the verifier’s messages. Additionally, we have  $c_0 = \emptyset$  in all protocols we consider, so that we will only consider  $(2r + 1)$ -message protocols (where the prover sends the first and last message).

### 2.3 Non-Interactive Arguments in the ROM

In practice, we often use the *Fiat-Shamir transform* (see Sect. 2.4) to compile public-coin interactive arguments into their *non-interactive* versions, in a model where both parties have black-box access to a *random oracle*, i.e. a uniformly sampled function  $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ . For public-coin  $(2r + 1)$ -message interactive arguments with challenge spaces  $\text{Ch}_1, \dots, \text{Ch}_r$ , we will actually need  $r$  independent random oracles  $H_i : \{0, 1\}^* \rightarrow \text{Ch}_i$  with  $i \in [1, r]$ . For simplicity, we will denote these by a single random oracle  $H$ , and it will be clear from context which random oracle is being used in a given round.

**Definition 2.3.** *A non-interactive argument (NARG) in the ROM for a relation  $\mathcal{R} \subseteq \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^*$  is a tuple of algorithms  $\Pi = (\text{Setup}, \mathcal{P}, \mathcal{V})$ , with  $\mathcal{P}, \mathcal{V}$  having black-box access to a random oracle  $H$ , with the following syntax:*

- $\text{Setup}(\text{pp}_G) \rightarrow \text{pp}$  generates the public parameters,
- $\mathcal{P}^H(\text{pp}, x, w) \rightarrow \pi$  generates a proof given  $\text{pp}$  and an input-witness pair  $(x, w)$ ,
- $\mathcal{V}^H(\text{pp}, x, \pi) \rightarrow \{0, 1\}$  checks if proof  $\pi$  is valid for  $\text{pp}$  and input  $x$ .

We define the following properties of NARGs:

- **Completeness.** For every adversary  $\mathcal{A}$ ,

$$\Pr \left[ \begin{array}{l} (\text{pp}, x, w) \notin \mathcal{R} \vee \\ \mathcal{V}^H(\text{pp}, x, \pi) = 1 \end{array} : \begin{array}{l} \text{pp} \leftarrow \text{Setup}(\text{pp}_G) \\ (x, w) \leftarrow \mathcal{A}^H(\text{pp}) \\ \pi \leftarrow \mathcal{P}^H(\text{pp}, x, w) \end{array} \right] = 1.$$

- **Knowledge Soundness.**  $\Pi$  is (adaptively) knowledge sound (KS) if there exists an extractor  $\mathcal{E}$  running in expected polynomial time such that for every PPT adversary  $\mathcal{P}^*$ , the following probability is negligible in  $\lambda$ :

$$\text{Adv}_{\Pi_{\text{FS}}, \mathcal{R}}^{\text{KS}}(\mathcal{E}, \mathcal{P}^*) := \left| \Pr[\text{KS}_{0, \Pi_{\text{FS}}}^{\mathcal{P}^*}(\lambda)] - \Pr[\text{KS}_{1, \Pi_{\text{FS}}, \mathcal{R}}^{\mathcal{E}, \mathcal{P}^*}(\lambda)] \right|.$$

The knowledge soundness games are defined in Fig. 1.



Game $\text{KS}_{0, \Pi_{\text{FS}}}^{\mathcal{P}^*}(\lambda)$	Game $\text{KS}_{1, \Pi_{\text{FS}}, \mathcal{R}}^{\mathcal{E}, \mathcal{P}^*}(\lambda)$
$\text{pp} \leftarrow \text{Setup}(\text{pp}_{\mathcal{G}})$	$\text{pp} \leftarrow \text{Setup}(\text{pp}_{\mathcal{G}})$
$(x, \pi) \leftarrow (\mathcal{P}^*)^{\text{H}}(\text{pp})$	$(x, \pi) \leftarrow (\mathcal{P}^*)^{\text{H}}(\text{pp})$
$b \leftarrow \mathcal{V}_{\text{FS}}^{\text{H}}(\text{pp}, x, \pi)$	$b \leftarrow \mathcal{V}_{\text{FS}}^{\text{H}}(\text{pp}, x, \pi)$
<b>return</b> $b$	$w \leftarrow \mathcal{E}^{\mathcal{P}^*}(\text{pp}, x, \pi)$
	<b>return</b> $b \wedge (\text{pp}, x, w) \in \mathcal{R}$

**Fig. 1.** Knowledge soundness security games. Here the extractor  $\mathcal{E}$  is given black-box access to  $\mathcal{P}^*$ . In particular,  $\mathcal{E}$  implements  $\text{H}$  for  $\mathcal{P}^*$  and can rewind  $\mathcal{P}^*$  to any point.

Game $\text{ZK}_{0, \Pi_{\text{FS}}, \mathcal{R}}^{\mathcal{D}, \mathcal{P}}(\lambda)$	Game $\text{ZK}_{1, \Pi_{\text{FS}}, \mathcal{R}}^{\mathcal{D}, \mathcal{S}}(\lambda)$
$\text{pp} \leftarrow \text{Setup}(\text{pp}_{\mathcal{G}})$	$\text{pp} \leftarrow \text{Setup}(\text{pp}_{\mathcal{G}})$
$b \leftarrow \mathcal{D}^{\text{H}(\cdot), \mathcal{P}'(\text{pp}, \cdot, \cdot)}(1^\lambda)$	$b \leftarrow \mathcal{D}^{\text{H}(\cdot), \mathcal{S}'(\text{pp}, \cdot, \cdot)}(1^\lambda)$
<b>return</b> $b$	<b>return</b> $b$
$\mathcal{P}'(\text{pp}, x, w)$	$\mathcal{S}'(\text{pp}, x, w)$
<b>if</b> $(\text{pp}, x, w) \notin \mathcal{R}$ <b>then return</b> $\perp$	<b>if</b> $(\text{pp}, x, w) \notin \mathcal{R}$ <b>then return</b> $\perp$
<b>else return</b> $\mathcal{P}(\text{pp}, x, w)$	<b>else return</b> $\mathcal{S}^{\text{RePro}}(\text{pp}, x)$

**Fig. 2.** Zero-knowledge security games. Here the simulator  $\mathcal{S}$  gets access to a RePro oracle that on input  $(a, b)$  reprograms  $\text{H}(a) := b$ .

We define zero-knowledge in a model where the random oracle is *explicitly-programmable* [58] by the simulator. Here, the simulator  $\mathcal{S}$  can reprogram the random oracle  $\text{H}$ , and this modified oracle is provided to the distinguisher.

**Definition 2.4 (Zero-Knowledge).**  $\Pi$  satisfies (statistical) unbounded non-interactive zero-knowledge (NIZK) if there exists a PPT simulator  $\mathcal{S}$  such that for  $\text{pp} \leftarrow \text{Setup}(\text{pp}_{\mathcal{G}})$  and any unbounded distinguisher  $\mathcal{D}$ , the following probability is negligible in  $\lambda$ :

$$\text{Adv}_{\Pi_{\text{FS}}, \mathcal{R}}^{\text{ZK}}(\mathcal{S}, \mathcal{D}) := \left| \Pr \left[ \text{ZK}_{0, \Pi_{\text{FS}}, \mathcal{R}}^{\mathcal{D}, \mathcal{P}}(\lambda) \right] - \Pr \left[ \text{ZK}_{1, \Pi_{\text{FS}}, \mathcal{R}}^{\mathcal{D}, \mathcal{S}}(\lambda) \right] \right|.$$

The zero-knowledge games are defined in Fig. 2.

### 2.4 The Fiat-Shamir Transformation

We define the Fiat-Shamir transform [25], which removes interaction from any public-coin interactive argument.

**Definition 2.5 (Fiat-Shamir Transformation).** Let  $\Pi = (\text{Setup}, \mathcal{P}, \mathcal{V})$  be a public-coin  $(2r + 1)$ -message interactive argument of knowledge. Denote the transcript as  $\text{tr} = (a_1, c_1, \dots, a_r, c_r, a_{r+1})$ . The Fiat-Shamir transformation turns  $\Pi$  into a non-interactive protocol  $\Pi_{\text{FS}}$  in the ROM, where:

- $\text{Setup}_{\mathcal{P}_{\text{FS}}}(\text{pp}_{\mathcal{G}})$  is the same as  $\text{Setup}(\text{pp}_{\mathcal{G}})$ ,
- the prover  $\mathcal{P}_{\text{FS}}$ , on input  $(\text{pp}, x, w)$ , invokes  $\mathcal{P}(x, w)$ , and instead of asking the verifier for challenge  $c_i$  in round  $i$ , queries the random oracle to get

$$c_i = \text{H}(\text{pp}, x, a_1, \dots, a_i) \text{ for all } i = 1, \dots, r.$$

$\mathcal{P}_{\text{FS}}$  then outputs a non-interactive proof  $\pi = (a_1, \dots, a_r, a_{r+1})$ .

- the verifier  $\mathcal{V}_{\text{FS}}$ , on input  $(\text{pp}, x, \pi)$ , derives challenges  $c_i$ 's by querying the random oracle as  $\mathcal{P}_{\text{FS}}$  does, then runs  $\mathcal{V}(\text{pp}, x, (a_1, c_1, \dots, a_r, c_r, a_{r+1}))$  and outputs what  $\mathcal{V}$  outputs.

For all protocols  $\Pi$  considered in this paper, it is clear that both  $\Pi$  and  $\Pi_{\text{FS}}$  satisfy (perfect) completeness. Furthermore,  $\Pi_{\text{FS}}$  satisfies knowledge soundness if  $\Pi$  is (computationally) special sound (see Sect. 4). For zero-knowledge, we have a canonical simulator  $\mathcal{S}_{\text{FS}}$  for  $\Pi_{\text{FS}}$  based on any HVZK simulator  $\mathcal{S}$  for  $\Pi$ .

**Definition 2.6 (Canonical Simulator).** Let  $\Pi$  be a public-coin interactive argument with HVZK simulator  $\mathcal{S}$ . Define the canonical simulator  $\mathcal{S}_{\text{FS}}$  for  $\Pi_{\text{FS}}$  to be an algorithm that on input  $(\text{pp}, x)$  runs  $\mathcal{S}(\text{pp}, x)$  to get a transcript  $\text{tr} = (a_1, c_1, \dots, a_r, c_r, a_{r+1})$ , then reprogram  $\text{H}(\text{pp}, x, a_1, \dots, a_i) := c_i$  for all  $i \in [r]$ .

*Remark 2.7.* It can be shown that  $\mathcal{S}_{\text{FS}}$  is a NIZK simulator for  $\Pi_{\text{FS}}$  if  $\mathcal{S}$  is an HVZK simulator and the fact that the first message  $a_1$  has sufficient min-entropy [24, 30]. Looking ahead, given any simulator  $\mathcal{S}$  for  $\Pi_{\text{FS}}$ , to show that it is a NIZK simulator, it suffices to show that  $\mathcal{S}$  produces indistinguishable transcripts  $\text{tr} = (a_1, c_1, \dots, a_{r+1})$  from honestly generated transcripts, and that the first message  $a_1$  has sufficient min-entropy.

### 3 Simulation Extractability

We define the central notion of our work, simulation extractability (SIM-EXT), which requires that extractability holds even when the malicious prover is given access to simulated proofs. SIM-EXT implies adaptive (knowledge) soundness and non-malleability for the proof system [30, 43, 50], and allows building secure signatures of knowledge via standard transforms [19, 39].

**Definition 3.1 (Simulation Extractability).** Let  $\Pi = (\text{Setup}, \mathcal{P}, \mathcal{V})$  be a public-coin zero-knowledge interactive argument for relation  $\mathcal{R}$  with associated NIZK  $\Pi_{\text{FS}} = (\text{Setup}, \mathcal{P}_{\text{FS}}, \mathcal{V}_{\text{FS}})$ . We say  $\Pi_{\text{FS}}$  satisfies simulation extractability (SIM-EXT) with respect to a simulator  $\mathcal{S}$  if there exists an efficient simulator-extractor  $\mathcal{E}$  such that for every PPT adversary  $\mathcal{P}^*$ , the following probability is negligible in  $\lambda$ :

$$\text{Adv}_{\Pi_{\text{FS}}, \mathcal{R}}^{\text{SIM-EXT}}(\mathcal{S}, \mathcal{E}, \mathcal{P}^*, \lambda) := \left| \Pr[\text{SIM-EXT}_{0, \Pi_{\text{FS}}}^{\mathcal{S}, \mathcal{P}^*}(\lambda)] - \Pr[\text{SIM-EXT}_{1, \Pi_{\text{FS}}, \mathcal{R}}^{\mathcal{E}, \mathcal{S}, \mathcal{P}^*}(\lambda)] \right|.$$

Games  $\text{SIM-EXT}_0$  and  $\text{SIM-EXT}_1$  are defined in Fig. 3.

Game $\text{SIM-EXT}_{0, \Pi_{\text{FS}}}^{\mathcal{S}, \mathcal{P}^*}(\lambda)$	Game $\text{SIM-EXT}_{1, \Pi_{\text{FS}}, \mathcal{R}}^{\mathcal{E}, \mathcal{S}, \mathcal{P}^*}(\lambda)$
$\text{pp} \leftarrow \text{Setup}(\text{pp}_{\mathcal{G}})$	$\text{pp} \leftarrow \text{Setup}(\text{pp}_{\mathcal{G}})$
$(x, \pi) \leftarrow (\mathcal{P}^*)^{\text{H}, \mathcal{S}}(\text{pp})$	$(x, \pi) \leftarrow (\mathcal{P}^*)^{\text{H}, \mathcal{S}}(\text{pp})$
$b \leftarrow \mathcal{V}_{\text{FS}}^{\text{H}' }(\text{pp}, x, \pi)$	$b \leftarrow \mathcal{V}_{\text{FS}}^{\text{H}' }(\text{pp}, x, \pi)$
<b>return</b> $b \wedge (x, \pi) \notin \mathcal{Q}_{\text{Sim}}$	$w \leftarrow \mathcal{E}^{\mathcal{P}^*}(\text{pp}, x, \pi)$
	<b>return</b> $b \wedge (x, \pi) \notin \mathcal{Q}_{\text{Sim}} \wedge (\text{pp}, x, w) \in \mathcal{R}$

**Fig. 3.** SIM-EXT security games. In both games,  $\mathcal{S}$  returns a proof  $\pi$  upon an input  $x$  (and may reprogram the random oracle), while  $\mathcal{Q}_{\text{Sim}}$  records all pairs  $(x, \pi)$  queried by  $\mathcal{P}^*$ .  $\text{H}'$  denotes the modified RO after all proof simulation queries.  $\mathcal{E}$  is given black-box access to  $\mathcal{P}^*$ ; in particular, it implements  $\text{H}$  and  $\mathcal{S}$  for  $\mathcal{P}^*$  and can rewind  $\mathcal{P}^*$  to any point in its execution (with same initial randomness).

We will state an adaptation of the results in [29], which establishes a general theorem about simulation extractability. In particular, the authors of [29] define the notion of a  $k$ -zero-knowledge simulator that only needs to reprogram the random oracle in round  $k$ . Similarly, they define a property of  $k$ -unique response, which roughly states that the malicious prover’s responses are uniquely determined after round  $k$ . Together, these two properties (for the same  $k$ ) along with knowledge soundness will be enough to show simulation extractability.

**Definition 3.2 ( $k$ -Zero-Knowledge).** Let  $\Pi = (\text{Setup}, \mathcal{P}, \mathcal{V})$  be a  $(2r + 1)$ -message public-coin interactive argument with HVZK simulator  $\mathcal{S}$ , and  $k \in [1, r]$ . Let  $\Pi_{\text{FS}}$  be its associated FS-transformed NIZK. We say  $\Pi_{\text{FS}}$  satisfies (perfect)  $k$ -zero-knowledge ( $k$ -ZK) if there exists a zero-knowledge simulator  $\mathcal{S}_{\text{FS}, k}$  that only needs to program the random oracle in round  $k$ , and whose output is identically distributed to that of honestly generated proofs.

**Definition 3.3 ( $k$ -Unique Response).** Let  $\Pi = (\text{Setup}, \mathcal{P}, \mathcal{V})$  be a  $(2r + 1)$ -message public-coin interactive argument, with  $\Pi_{\text{FS}}$  its associated FS-transformed NARG and  $k \in [0, r]$ . We say  $\Pi_{\text{FS}}$  satisfies  $k$ -unique response ( $k$ -UR) if for all PPT adversaries  $\mathcal{A}$ , the following probability (defined with respect to the game in Fig. 4) is negligible in  $\lambda$ :

$$\text{Adv}_{\Pi_{\text{FS}}}^{k\text{-UR}}(\mathcal{A}) := \Pr \left[ k\text{-UR}_{\Pi_{\text{FS}}}^{\mathcal{A}}(\lambda) \right].$$

When  $k = 0$ , we say that  $\Pi_{\text{FS}}$  has (computationally) unique proofs.

We now state a key theorem that relates SIM-EXT to these properties; it is similar to the SIM-EXT theorem given in [29], with SRS update oracles removed. We give the proof in the full version.

**Theorem 3.4.** Let  $\Pi_{\text{FS}}$  be a Fiat-Shamir compiled non-interactive argument for relation  $\mathcal{R}$  from a  $(2r + 1)$ -message public-coin interactive argument  $\Pi$ . Assume  $\Pi_{\text{FS}}$  satisfies KS, has a perfect  $k$ -ZK simulator  $\mathcal{S}_{\text{FS}, k}$  for  $k \in [1, r]$ , and satisfies  $k$ -UR (for the same  $k$ ). Then  $\Pi_{\text{FS}}$  satisfies SIM-EXT.

**Game**  $k\text{-UR}_{\Pi_{\text{FS}}}^{\mathcal{A}}(\lambda)$

---

$\text{pp} \leftarrow \text{Setup}(1^\lambda, \text{pp}_{\mathcal{G}})$

$(x, \pi, \pi', c) \leftarrow \mathcal{A}^{\text{H}}(\text{pp})$

$b \leftarrow \mathcal{V}_{\text{FS}}^{\text{H}[(\text{pp}, x, \pi|_k) \mapsto c]}(\text{pp}, x, \pi) = 1$

$b' \leftarrow \mathcal{V}_{\text{FS}}^{\text{H}[(\text{pp}, x, \pi'|_k) \mapsto c]}(\text{pp}, x, \pi') = 1$

**return**  $b \wedge b' \wedge \pi \neq \pi' \wedge \pi|_k = \pi'|_k$

**Fig. 4.** Security game for  $k$ -unique response. Here  $\text{H}[(\text{pp}, x, \pi|_k) \mapsto c]$  denotes the random oracle where the input  $(\text{pp}, x, \pi|_k)$  is reprogrammed to output  $c$ .

Concretely, let  $\mathcal{E}$  be a KS extractor for  $\Pi_{\text{FS}}$ . There exists a SIM-EXT simulator-extractor  $\mathcal{E}_{\text{SE}}$  for  $\Pi_{\text{FS}}$  such that for every PPT prover  $\mathcal{P}^*$  against  $\Pi_{\text{FS}}$  that makes at most  $q_{\text{H}}$  random oracle queries and  $q_{\text{Sim}}$  simulation queries, there exists another PPT prover  $\mathcal{P}_{\text{KS}}^*$  against KS and PPT adversary  $\mathcal{A}$  against  $k\text{-UR}$  such that  $\text{Adv}_{\Pi_{\text{FS}}, \mathcal{R}}^{\text{SIM-EXT}}(\mathcal{S}_{\text{FS}, k}, \mathcal{E}_{\text{SE}}, \mathcal{P}^*) \leq \text{Adv}_{\Pi_{\text{FS}}, \mathcal{R}}^{\text{KS}}(\mathcal{E}, \mathcal{P}_{\text{KS}}^*) + \text{Adv}_{\Pi_{\text{FS}}}^{k\text{-UR}}(\mathcal{A}) + 2/|\text{Ch}_k|$ . Here  $\text{Ch}_k$  is the challenge set in round  $k$ . Furthermore, both  $\mathcal{P}_{\text{KS}}^*$  and  $\mathcal{A}$  make at most  $q_{\text{H}}$  random oracle queries; their runtime is roughly equal to  $\mathcal{P}^*$ 's runtime plus  $q_{\text{Sim}}$  invocations of  $\mathcal{S}_{\text{FS}, k}$ .  $\mathcal{E}_{\text{SE}}$  is nearly as efficient as  $\mathcal{E}$ .

### 4 Tree of Transcripts and Special Soundness

In this section, we show how to establish knowledge soundness (KS) of a FS-transformed protocol  $\Pi_{\text{FS}}$  based on the *computational special soundness* of the interactive protocol  $\Pi$ . The key is to construct an efficient *tree builder*  $\mathcal{TB}$  that, given oracle access to a malicious prover  $\mathcal{P}^*$  for  $\Pi_{\text{FS}}$ , outputs a suitable *tree of accepting transcripts*, upon which a valid witness can be extracted.

**Definition 4.1 (Tree of Transcripts).** Let  $\Pi$  be a  $(2r + 1)$ -message public-coin interactive argument for a relation  $\mathcal{R}$ , with challenge spaces  $\text{Ch}_1, \dots, \text{Ch}_r$ . Given  $\mathbf{n} = (n_1, \dots, n_r) \in \mathbb{N}^r$  and  $\phi = (\phi_1, \dots, \phi_r)$  with  $\phi_i : \text{Ch}_i^{n_i} \rightarrow \{0, 1\}$  for  $i \in [r]$ , we say that  $\mathcal{T}$  is a  $(\phi, \mathbf{n})$ -tree of accepting transcripts for  $\text{pp}$  if:

1.  $\mathcal{T}$  is a tree of depth  $r + 1$ ,
2. For each  $i \in [r + 1]$ , each vertex at depth  $i$  is labeled with a prover's  $i$ -th message  $a_i$ , and if  $i \leq r$ , has exactly  $n_i$  outgoing edges to its children, with each edge labeled with a verifier's  $i$ -th challenge  $c_{i,1}, \dots, c_{i,n_i}$  satisfying  $\phi_i(c_{i,1}, \dots, c_{i,n_i}) = 1$ . Additionally, the root's label is prepended with  $x$  (so the label becomes  $(x, a_1)$ ),
3. The labels on any root-to-leaf path form a valid input-transcript pair  $(x, \text{tr})$ .

We additionally define  $\mathcal{T}$  to be accepting with respect to a input-transcript pair  $(x, \text{tr})$  if  $(x, \text{tr})$  corresponds to the left-most path of  $\mathcal{T}$ . We define a predicate  $\text{IsAccepting}((\phi, \mathbf{n}), \text{pp}, x, (\pi, \cdot), \mathcal{T})$  to check whether  $\mathcal{T}$  is a  $(\phi, \mathbf{n})$ -tree of accepting transcripts for  $\text{pp}$  and  $x$ , and optionally  $\pi$ .

The usual definition of a tree of accepting transcripts [2, 14] has  $\phi_i$  be the predicate that the  $i$ -th challenges  $c_{i,1}, \dots, c_{i,n_i}$ , coming from a vertex at depth  $i$ , are distinct (we call this the *distinctness predicate*). In that case, we will also abbreviate  $\mathcal{T}$  as a  $\mathbf{n}$ -tree of accepting transcripts. However, we will need to consider more general *partition predicates* in our proofs of knowledge soundness for Spartan and Bulletproofs.

**Definition 4.2 (Partition Predicate).** Let  $\text{Ch} = \text{Ch}^{(1)} \sqcup \text{Ch}^{(2)} \dots \sqcup \text{Ch}^{(C)}$  be a partition  $\mathcal{P}$  of a set  $\text{Ch}$  into  $C$  blocks. We assume the partition is efficient, i.e. given an index  $i \in [C]$ , we can enumerate the set  $\text{Ch}^{(i)}$  in polynomial time. For  $n \in \mathbb{N}$ , we define the corresponding partition predicate  $\phi_{\mathcal{P},n} : \text{Ch}^n \rightarrow \{0, 1\}$  to be  $\phi_{\mathcal{P},n}(c_1, \dots, c_n) = 1$  if and only if  $c_1, \dots, c_n$  belong in distinct blocks of  $\text{Ch}$ .

*Remark 4.3.* Looking ahead, we will consider the following partition predicates:

- When  $\text{Ch} = \mathbb{F}^*$  is partitioned into  $\{x, -x\}$  for all  $x$ . We abbreviate this predicate into the number  $n$  of challenges as  $n_{\pm}$ .
- When  $\text{Ch} = \mathbb{F}^2$  is partitioned into  $\{c \cdot x \mid c \in \mathbb{F}^*\}$  for all  $x \in \{(0, 0), (0, 1)\} \cup \{(1, a) \mid a \in \mathbb{F}\}$  (this implies linear independence between two vectors). We abbreviate this predicate into the number  $n$  of challenges as  $n_{\text{ij}}$ .

We now state a theorem asserting the existence of an efficient tree-builder that can generate  $(\phi, \mathbf{n})$ -trees of accepting transcripts, where  $\phi$  consists of partition predicates as defined above. In the full version, we give a proof of this theorem along with a comparison of our tree-builder with that of Wikström [59].

**Theorem 4.4 (Efficient Tree Builder).** Let  $\Pi$  be a  $(2r + 1)$ -message public-coin interactive argument with challenge spaces  $\text{Ch}_1, \dots, \text{Ch}_r$ . Consider any efficiently decidable partition  $\text{Ch}_i = \sqcup_{j=1}^{C_i} \text{Ch}_{i,j}$  with minimum partition size  $C = \min_i C_i$ , and let  $\phi = (\phi_1, \dots, \phi_r)$  be the corresponding partition predicate. Consider any  $\mathbf{n} = (n_1, \dots, n_r) \in \mathbb{N}^r$  with  $N = \prod_{i=1}^r n_i$ .

There exists a probabilistic algorithm  $\mathcal{TB}$  for  $\Pi_{\text{FS}}$  with the following guarantees: given oracle access to a malicious prover  $\mathcal{P}^*$  for  $\Pi_{\text{FS}}$  with success probability  $\epsilon(\mathcal{P}^*) := \Pr[\text{KS}_{0, \Pi_{\text{FS}}}^{\mathcal{P}^*}]$ ,  $\mathcal{TB}$  wins the tree-building game  $\text{TreeBuild}_{\Pi_{\text{FS}}, (\phi, \mathbf{n})}^{\mathcal{TB}, \mathcal{P}^*}$  (shown in Fig. 5) with probability at least

$$\Pr \left[ \text{TreeBuild}_{\Pi_{\text{FS}}, (\phi, \mathbf{n})}^{\mathcal{TB}, \mathcal{P}^*} \right] \geq \epsilon(\mathcal{P}^*) - \frac{Q(Q-1)/2 + (Q+1)(\sum_{i=1}^r n_i - r)}{C}.$$

Furthermore,  $\mathcal{TB}$  makes in expectation at most  $(Q + 1)(N - 1) + 1$  rewinding calls to  $\mathcal{P}^*$ , where  $Q$  is an upper bound on the number of RO queries of  $\mathcal{P}^*$ .

We now define computational special soundness, which stipulates the existence of a tree-extraction procedure  $\mathcal{TE}$  that, given an appropriate tree of accepting transcripts produced by an efficient adversary, outputs a witness with high probability.

Game $\text{TreeBuild}_{\Pi_{\text{FS}},(\phi,\mathbf{n})}^{\mathcal{TB},\mathcal{P}^*}(\lambda)$	Game $\text{SS}_{\Pi,\mathcal{R},(\phi,\mathbf{n})}^{\mathcal{TE},\mathcal{A}}(\lambda)$
$\text{pp} \leftarrow \text{Setup}(\text{pp}_{\mathcal{G}})$	$\text{pp} \leftarrow \text{Setup}(\text{pp}_{\mathcal{G}})$
$(x, \pi) \leftarrow (\mathcal{P}^*)^{\text{H}}(\text{pp})$	$(x, \mathcal{T}) \leftarrow \mathcal{A}(\text{pp})$
$\mathcal{T} \leftarrow \mathcal{TB}^{\mathcal{P}^*}(\text{pp}, x, \pi)$	$w \leftarrow \mathcal{TE}(\text{pp}, x, \mathcal{T})$
<b>return</b> $(\mathcal{V}^{\text{H}}(\text{pp}, x, \pi) = 1) \wedge$	<b>return</b> $(\text{pp}, x, w) \notin \mathcal{R} \wedge$
$\text{IsAccepting}((\phi, \mathbf{n}), \text{pp}, x, \pi, \mathcal{T})$	$\text{IsAccepting}((\phi, \mathbf{n}), \text{pp}, x, \mathcal{T})$

**Fig. 5.** Games for tree-building and special soundness. Here the tree-builder  $\mathcal{TB}$  is given black-box access to  $\mathcal{P}^*$ . In particular,  $\mathcal{TB}$  implements  $\text{H}$  for  $\mathcal{P}^*$  and can rewind  $\mathcal{P}^*$  to any point in its execution.

**Definition 4.5 (Special Soundness).** *Let  $\Pi$  be a  $(2r + 1)$ -message public-coin interactive argument for a relation  $\mathcal{R}$  with challenge spaces  $\text{Ch}_1, \dots, \text{Ch}_r$ . For any  $\mathbf{n} = (n_1, \dots, n_r) \in \mathbb{N}^r$  and  $\phi = (\phi_1, \dots, \phi_r)$  with  $\phi_i : \text{Ch}_i^{n_i} \rightarrow \{0, 1\}$ , we say  $\Pi$  is  $(\phi, \mathbf{n})$ -computational special sound if there exists a PPT tree-extraction algorithm  $\mathcal{TE}$  such that for all EPT adversary  $\mathcal{A}$ , the following probability is negligible in  $\lambda$ :*

$$\text{Adv}_{\Pi,\mathcal{R},(\phi,\mathbf{n})}^{\text{SS}}(\mathcal{TE}, \mathcal{A}) := \Pr \left[ \text{SS}_{\Pi,\mathcal{R},(\phi,\mathbf{n})}^{\mathcal{TE},\mathcal{A}}(\lambda) \right].$$

The special soundness game is shown in Fig. 5. We say  $\Pi$  is computational special sound (SS) if it is  $(\phi, \mathbf{n})$ -computational special sound for some  $\phi$  and  $\mathbf{n}$ .

Using Theorem 4.4 and Definition 4.5, we get the following consequence that computational special soundness for an interactive protocol implies knowledge soundness for its non-interactive version.

**Lemma 4.6.** *Let  $\Pi$  be a  $(2r + 1)$ -message public-coin interactive argument that is  $(\phi, \mathbf{n})$ -computational special sound with tree extractor  $\mathcal{TE}$ , where  $\mathbf{n} = (n_1, \dots, n_r) \in \mathbb{N}^r$  and  $\phi$  is a partition predicate with minimum partition size  $C$ . Then  $\Pi_{\text{FS}}$  satisfies knowledge soundness. Concretely, there exists an EPT extractor  $\mathcal{E}$  such that for every PPT adversary  $\mathcal{P}^*$  against KS making at most  $Q$  random oracle calls, there exists an EPT adversary  $\mathcal{A}$  against SS such that*

$$\text{Adv}_{\Pi_{\text{FS}},\mathcal{R}}^{\text{KS}}(\mathcal{E}, \mathcal{P}^*) \leq \frac{Q(Q - 1)/2 + (Q + 1) (\sum_{i=1}^r n_i - r)}{C} + \text{Adv}_{\Pi,(\phi,\mathbf{n})}^{\text{SS}}(\mathcal{TE}, \mathcal{A}).$$

Both  $\mathcal{E}$  and  $\mathcal{A}$  runs in expected time that is at most  $O(Q \cdot N)$  the runtime of  $\mathcal{P}^*$ .

*Proof.* Our proof goes through a sequence of hybrids.  $\text{Hyb}_0$  is the game  $\text{KS}_{0,\Pi_{\text{FS}}}^{\mathcal{P}^*}$ .  $\text{Hyb}_1$  is the same as  $\text{Hyb}_0$ , except we also run  $\mathcal{TB}^{\mathcal{P}^*}(\text{pp}, x, \pi) \rightarrow \mathcal{T}$  and output 0 if  $\mathcal{T}$  is not a  $(\phi, \mathbf{n})$ -tree of accepting transcripts with respect to  $(\text{pp}, x, \pi)$ . Note that  $\text{Hyb}_1$  is the same as the game  $\text{TreeBuild}_{\Pi_{\text{FS}},(\phi,\mathbf{n})}^{\mathcal{TB},\mathcal{P}^*}$ . Using Theorem 4.4, we get

$$|\Pr[\text{Hyb}_0] - \Pr[\text{Hyb}_1]| \leq \frac{Q(Q - 1)/2 + (Q + 1) (\sum_{i=1}^r n_i - r)}{C}.$$

We define  $\text{Hyb}_2$  to be the same as  $\text{Hyb}_1$ , except we also run  $\mathcal{TE}(\text{pp}, x, \mathcal{T}) \rightarrow w$  and output 0 if  $(\text{pp}, x, w) \notin \mathcal{R}$ . We define the extractor  $\mathcal{E}$  to be as follows: run  $\mathcal{TB}^{\mathcal{P}^*}(\text{pp}, x, \pi) \rightarrow \mathcal{T}$  to obtain a tree of accepting transcripts, then run  $\mathcal{TE}(\text{pp}, x, \mathcal{T}) \rightarrow w$  to obtain a witness. By definition of  $\mathcal{E}$ , we can see that  $\text{Hyb}_2$  is the same as the game  $\text{KS}_{1, \Pi_{\mathbb{F}_5}, \mathcal{R}}^{\mathcal{E}, \mathcal{P}^*}$ .

We now claim that there exists an adversary  $\mathcal{A}$  against SS such that

$$|\Pr[\text{Hyb}_1] - \Pr[\text{Hyb}_2]| \leq \text{Adv}_{\Pi, (\phi, \mathbf{n})}^{\text{SS}}(\mathcal{TE}, \mathcal{A}).$$

We define  $\mathcal{A}$  to be as follows: given oracle access to  $\mathcal{P}^*$ ,  $\mathcal{A}$  runs  $(\mathcal{P}^*)^{\text{H}}(\text{pp}) \rightarrow (x, \pi)$  by simulating H for  $\mathcal{P}^*$ , then runs  $\mathcal{TB}^{\mathcal{P}^*}(\text{pp}, x, \pi) \rightarrow \mathcal{T}$ , and outputs  $(x, \mathcal{T})$ . It is then straightforward to argue that  $\text{Hyb}_2$  returns 0 while  $\text{Hyb}_1$  returns 1 precisely when  $\mathcal{A}$  wins in SS.  $\square$

## 5 Simulation Extractability of Spartan

In this section, we use our general theorems to prove SIM-EXT of Spartan [54], a transparent zkSNARKs with security based on the discrete log assumption. [54] presents two version of Spartan, one with a linear verifier (called Spartan-NIZK) and one with a sublinear verifier (called Spartan-SNARK) achieved via encoding the R1CS matrices with a *sparse multilinear polynomial commitment*.

### 5.1 Spartan Protocols

We first describe the two variants of Spartan. Note that in a slight abuse of terminology, we will use Spartan-NIZK and Spartan-SNARK to refer to the interactive versions of their respective protocols. When we wish to refer specifically to the non-interactive versions, we will write  $\text{Spartan-NIZK}_{\mathbb{F}_5}$  and  $\text{Spartan-SNARK}_{\mathbb{F}_5}$ .

**Definition 5.1 (R1CS).** A R1CS instance is a tuple  $(\mathbb{F}, A, B, C, m, n, \text{io})$  where  $A, B, C \in \mathbb{F}^{m \times m}$  each with at most  $n = \Omega(m)$  non-zero entries, and  $m \geq |\text{io}| + 1$ . A R1CS witness is a vector  $w \in \mathbb{F}^{m - |\text{io}| - 1}$  such that if  $Z = (\text{io}, 1, w)$ , then  $(A \cdot Z) \circ (B \cdot Z) = C \cdot Z$ .

Spartan makes further assumptions on the R1CS instances, namely that  $m = 2^\mu, n = 2^\nu$  are powers of two, and  $|\text{io}| + 1 = |w| = m/2$ .

*Key ideas.* Both the NIZK and SNARK variants of Spartan prove satisfiability of R1CS instances using roughly the same ideas we now outline. See Fig. 6 for a protocol description. It uses the following sub-protocols (description in full version):

1. The Pedersen commitment scheme  $\mathbf{g}^{\mathbf{a}} \cdot h^\omega \leftarrow \text{Commit}((n, \mathbf{g}, h), \mathbf{a}; \omega)$ .
2. Four  $\Sigma$ -protocols sharing the same setup:
  - (a) OpenPf to prove knowledge of a commitment  $C = g^x \cdot h^\omega$ ,
  - (b) EqPf to prove equality of two commitments  $C_1 = g^x \cdot h^{\omega_1}, C_2 = g^x \cdot h^{\omega_2}$ ,
  - (c) ProdPf to prove that three commitments  $C_{v_1}, C_{v_2}, C_{v_3}$  satisfy  $v_1 \cdot v_2 = v_3$ ,

- (d) DotProdPf to prove that a multi-commitment  $C_{\mathbf{x}}$  and a commitment  $C_y$  satisfy  $y = \langle \mathbf{x}, \mathbf{a} \rangle$  for a public vector  $\mathbf{a}$ ,
- 3. A  $(\mu+1)$ -round public-coin interactive protocol  $\text{PC}_{\text{Multi}}.\text{Open}$  for proving polynomial evaluations of any multilinear polynomial  $p(X_1, \dots, X_\mu)$ .
- 4. Additionally, in the case of **Spartan-SNARK**, we also need  $\text{PC}_{\text{SparseMulti}}.\text{Open}$  for proving evaluations of *sparse* multilinear polynomials  $\tilde{A}, \tilde{B}, \tilde{C}$ .

At a high level, the main idea of **Spartan** is to reduce the satisfiability of the given R1CS instance to a claim that can be verified via sumcheck. To do this, the matrices  $A, B, C$  are interpreted as functions  $\{0, 1\}^\mu \times \{0, 1\}^\mu \rightarrow \mathbb{F}$ , and similarly  $Z : \{0, 1\}^\mu \rightarrow \mathbb{F}$ , by writing the indices as their binary representations. We then take the *multilinear extension*  $\tilde{A}, \tilde{B}, \tilde{C}, \tilde{Z}$  of these functions, and define the polynomial

$$\tilde{\mathcal{F}}_{\text{io}}(X) = \left( \sum_{y \in \{0,1\}^\mu} \tilde{A}(X, y) \cdot \tilde{Z}(y) \right) \cdot \left( \sum_{y \in \{0,1\}^\mu} \tilde{B}(X, y) \cdot \tilde{Z}(y) \right) - \left( \sum_{y \in \{0,1\}^\mu} \tilde{C}(X, y) \cdot \tilde{Z}(y) \right).$$

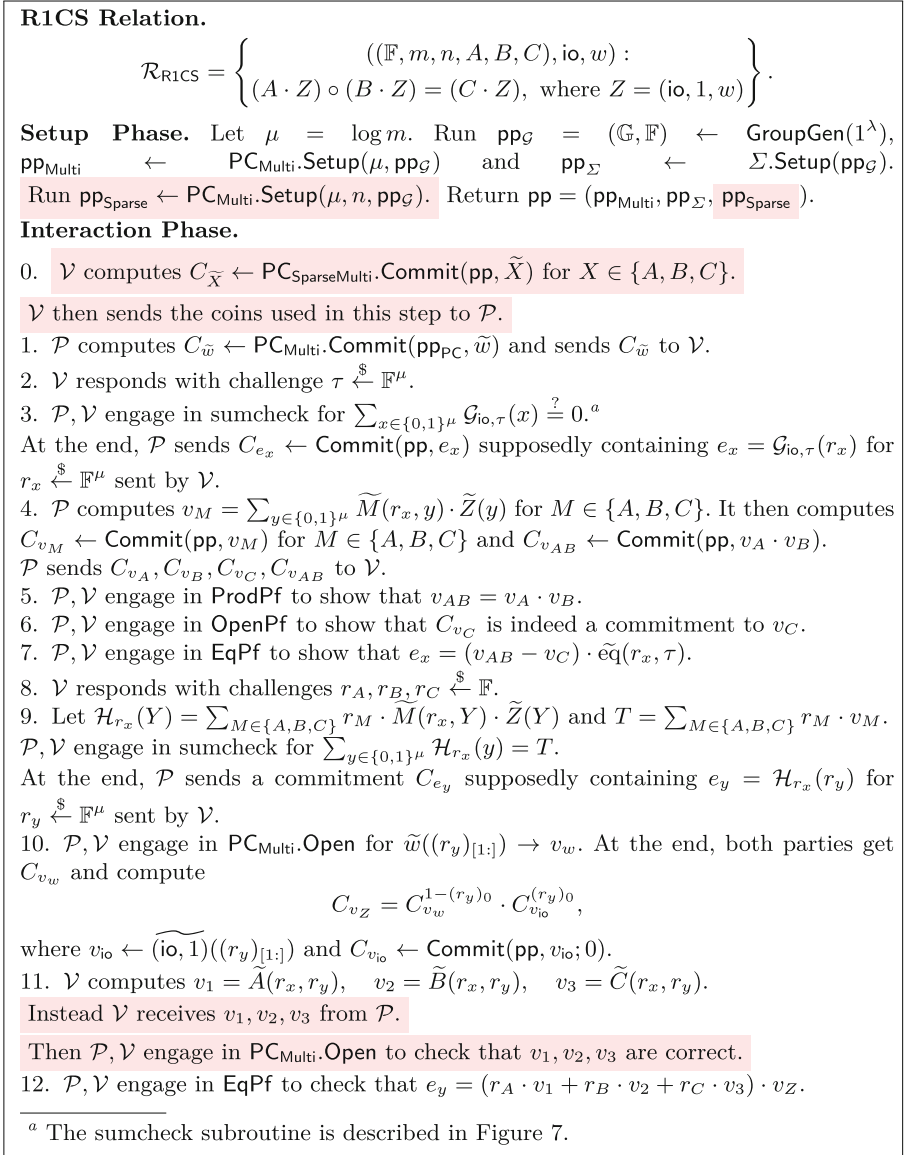
Note that  $\tilde{\mathcal{F}}_{\text{io}}(X)$  vanishes on  $\{0, 1\}^\mu$  if and only if the R1CS constraint is satisfied. Finally, we turn this vanishing condition into a sumcheck instance by defining  $\mathcal{G}_{\text{io}, \tau}(X) = \tilde{\mathcal{F}}_{\text{io}}(X) \cdot \tilde{\text{eq}}(X, \tau)$  for a random  $\tau \in \mathbb{F}^\mu$ , supplied by the verifier. The goal is then to prove that  $\sum_{y \in \{0,1\}^\mu} \mathcal{G}_{\text{io}, \tau}(y) = 0$ . The prover and verifier engage in sumcheck for this claim. The final step of sumcheck requires the verifier to evaluate  $\mathcal{G}_{\text{io}, \tau}$  at a random point  $r_x$ , but the verifier cannot do this itself; thus, the prover and verifier engage in another run of sumcheck (more precisely, three runs batched together with verifier randomness) to reduce the task of evaluating  $\mathcal{G}_{\text{io}, \tau}(r_x)$  to evaluating  $\tilde{A}, \tilde{B}, \tilde{C}$  all at  $(r_x, r_y)$ , and  $\tilde{Z}$  at  $r_y$ . In both **Spartan-NIZK** and **Spartan-SNARK**, the verifier gets a commitment to the evaluation of the witness, and is convinced the committed evaluation is correct via  $\text{PC}_{\text{Multi}}.\text{Open}$ . (Our analyses below assume  $\text{PC}_{\text{Multi}}$  is instantiated with HyraxPC [57].) In **Spartan-NIZK**, the verifier evaluates  $\tilde{A}, \tilde{B}, \tilde{C}$  itself; in **Spartan-SNARK**, the prover sends the verifier the evaluations and uses  $\text{PC}_{\text{SparseMulti}}.\text{Open}$ , a secondary proof protocol, to convince the verifier of their correctness.

We can compute the number of rounds of **Spartan-NIZK** to be  $r = 7\mu + 11$ . For **Spartan-SNARK**, the transcript is the same except for the verifier sending its commitments to  $\tilde{A}, \tilde{B}, \tilde{C}$  to the prover, the evaluations  $v_1, v_2, v_3$ , and the  $\mathcal{O}(\mu)$ -round transcript of  $\text{PC}_{\text{SparseMulti}}.\text{Open}$ . Thus, the transcript of **Spartan-SNARK** has  $\mathcal{O}(\mu)$  more rounds for evaluating  $\tilde{A}(r_x, r_y), \tilde{B}(r_x, r_y), \tilde{C}(r_x, r_y)$ .

## 5.2 SIM-EXT Analysis of Spartan-NIZK

Following Theorem 3.4, to prove that **Spartan-NIZK**<sub>FS</sub> satisfies SIM-EXT, we will need to show that it satisfies knowledge soundness (KS) along with  $k$ -ZK and  $k$ -UR for the same round  $k$ . By Lemma 4.6, knowledge soundness in turn depends on computational special soundness (SS) of the interactive protocol **Spartan-NIZK**. Our first set of results will be to establish SS of **Spartan-NIZK** through the following steps: (1) We first analyze the information-theoretic core





**Fig. 6.** Spartan-NIZK, with modifications for Spartan-SNARK in red (Color figure online).

**Sumcheck Sub-Protocol.** The sumcheck relation is  $\sum_{x \in \{0,1\}^\mu} p(x) = T$ , where  $p$  is a multivariate polynomial of individual degree at most  $d$ .  $\mathcal{V}$  is given a commitment  $C_p$  and a commitment  $C_T$ . The sumcheck subprotocol reduces this claim to the claim that  $p(r_x) \stackrel{?}{=} e_x$  for a random  $r_x \stackrel{\$}{\leftarrow} \mathbb{F}^\mu$  sampled randomly by  $\mathcal{V}$ , and some claimed value  $e_x \in \mathbb{F}$  available as a commitment  $C_{e_x}$  to  $\mathcal{V}$ .

Let  $e_0 = T$ . For  $i = 1, \dots, \mu$ :

1.  $\mathcal{P}$  computes the polynomial  $p_i(X) = \sum_{x \in \{0,1\}^{\mu-i}} P(r_1, \dots, r_{i-1}, X, x)$ , parse it as a vector of coefficients, then sends  $C_{p_i} \leftarrow \text{Commit}(\text{pp}, p_i; \omega_{p_i})$  to  $\mathcal{V}$ .
2.  $\mathcal{V}$  responds with challenge  $r_i \stackrel{\$}{\leftarrow} \mathbb{F}$ .
3.  $\mathcal{P}$  computes  $e_i = p_i(r_i)$ , then sends  $C_{e_i} \leftarrow \text{Commit}(\text{pp}, e_i; \omega_{e_i})$  to  $\mathcal{V}$ .
4.  $\mathcal{V}$  responds with challenges  $w_{i,1}, w_{i,2} \stackrel{\$}{\leftarrow} \mathbb{F}$ .
5.  $\mathcal{P}, \mathcal{V}$  compute  $\mathbf{a} = w_{i,1} \cdot (\mathbf{0}^k + \mathbf{1}^k) + w_{i,2} \cdot \mathbf{r}_i^k$  and  $C_{y_i} = C_{e_{i-1}}^{w_{i,1}} \cdot C_{e_i}^{w_{i,2}}$ . In addition,  $\mathcal{P}$  computes  $y_i = w_{i,1} \cdot e_{i-1} + w_{i,2} \cdot e_i$  and  $\omega_{y_i} = w_{i,1} \cdot \omega_{e_{i-1}} + w_{i,2} \cdot \omega_{e_i}$ .
6.  $\mathcal{P}, \mathcal{V}$  engage in  $\text{DotProdPf}(\text{pp}, (C_{p_i}, C_{y_i}, \mathbf{a}), (p_i, \omega_{p_i}, y_i, \omega_{y_i}))$ .

**Fig. 7.** Sumcheck Sub-Protocol

of Spartan-NIZK, which is obtained from the protocol by sending all polynomials and evaluations in the clear, and checking the equalities directly. We call this variant **Spartan-Core**. (2) We then analyze how to extract from the various commitments and subprotocols in **Spartan-NIZK** to recover **Spartan-Core**.

The soundness of **Spartan-Core** has been analyzed in [54].

**Lemma 5.2** ([54]). *Spartan-Core has soundness error  $\frac{6\mu+1}{|\mathbb{F}|}$ .*

Special soundness for  $\Sigma$ -protocols was analyzed in another previous work [57].

**Lemma 5.3** ([57]). *Let  $\Pi \in \{\text{OpenPf}, \text{EqPf}, \text{ProdPf}, \text{DotProdPf}\}$ . Then  $\Pi$  is 2-perfect special sound. Concretely, there exists a tree-extraction algorithm  $\mathcal{TE}_\Pi$  that can extract a valid witness for  $\Pi$  given any 2-tree of accepting transcripts.*

We also need to analyze special soundness of  $\text{PC}_{\text{Multi}}.\text{Open}$ . Note that while [57] introduced this protocol, they did not provide a concrete soundness result for it. The proof of the lemma below appears in the full version.

**Lemma 5.4.**  $\text{PC}_{\text{Multi}}.\text{Open}$  is  $\mathbf{n} = (\sqrt{m}, \underbrace{4_\pm, \dots, 4_\pm}_{\mu/2}, 2)$ -computational special sound. Concretely, there exists a tree-extraction algorithm  $\mathcal{TE}_{\text{PC}_{\text{Multi}}}$  such that for any EPT adversary  $\mathcal{A}$  against SS of  $\text{PC}_{\text{Multi}}.\text{Open}$ , there exists an EPT adversary  $\mathcal{B}$  against DL-REL, as efficient as  $\mathcal{A}$  and  $\mathcal{TE}_{\text{PC}_{\text{Multi}}}$  combined, such that

$$\text{Adv}_{\Pi, \mathbf{n}}^{\text{SS}}(\mathcal{TE}_{\text{PC}_{\text{Multi}}}, \mathcal{A}) \leq \text{Adv}_{\mathbb{G}, \sqrt{m}+2}^{\text{DL-REL}}(\mathcal{B}).$$

Our next step is to analyze the computational special soundness of the sumcheck subprotocol in Fig. 7. Since it is not strictly an interactive argument, we explicitly state the guarantees of the tree extractor.

**Lemma 5.5.** *There exists a tree extractor  $\mathcal{TE}_{\text{SC}}$  such that given a  $(1, 2_{\text{li}}, 2)^\mu$ -tree of accepting transcripts, produced by an adversary  $\mathcal{A}$ , for the sumcheck subprotocol, either outputs polynomials  $p_1(X), \dots, p_\mu(X)$  that satisfy the information-theoretic sumcheck protocol, or we can build an adversary  $\mathcal{B}$ , as efficient as  $\mathcal{TE}_{\text{SC}}$  and  $\mathcal{A}$  combined, against DL-REL.*

*Proof.* We will analyze a single iteration  $i \in [\mu]$  of the sumcheck subprotocol; all other iterations will follow the same reasoning. We construct a tree extractor  $\mathcal{TE}_{\text{SC}}$  that does the following for each iteration  $i \in [\mu]$ : given a  $(1, 2_{\text{li}}, 2)$ -tree of transcripts,

1. Run  $\mathcal{TE}_{\text{DotProdPf}}$  on each  $(1, 1, 2)$ -subtree to extract  $(p_i, \omega_{p_i}, y_i, \omega_{y_i})$ , where

$$C_{p_i} = \text{PC.Commit}(\text{pp}, p_i; \omega_{p_i}), \quad C_{y_i} = \text{Commit}(\text{pp}, y_i; \omega_{y_i}), \quad \langle p_i, \mathbf{a}_i \rangle = y_i,$$

and  $y_i$  is supposedly equal to  $w_{i,1} \cdot e_{i-1} + w_{i,2} \cdot e_i$ .

2. Given two pairs of linearly independent challenges  $(w_{i,1}, w_{i,2}), (w'_{i,1}, w'_{i,2})$ , with extracted witnesses  $(p_i, \omega_{p_i}, y_i, \omega_{y_i}), (p'_i, \omega'_{p_i}, y'_i, \omega'_{y_i})$  from the previous step, we first assert that  $\langle p_i, \omega_{p_i} \rangle = \langle p'_i, \omega'_{p_i} \rangle$ . If this assertion fails, then we have an adversary  $\mathcal{B}$  against DL-REL since  $C_{p_i} = \mathbf{g}^{p_i} \cdot h^{\omega_{p_i}} = \mathbf{g}^{p'_i} \cdot h^{\omega'_{p_i}}$ . Next, we can solve for  $e_{i-1}, e_i, \omega_{e_{i-1}}, \omega_{e_i}$  from the linear equations

$$\begin{cases} y_i = w_{i,1} \cdot e_{i-1} + w_{i,2} \cdot e_i \\ y'_i = w'_{i,1} \cdot e_{i-1} + w'_{i,2} \cdot e_i \end{cases} \quad \text{and} \quad \begin{cases} \omega_{y_i} = w_{i,1} \cdot \omega_{e_{i-1}} + w_{i,2} \cdot \omega_{e_i} \\ \omega'_{y_i} = w'_{i,1} \cdot \omega_{e_{i-1}} + w'_{i,2} \cdot \omega_{e_i} \end{cases}.$$

Recall that we also have  $\langle p_i, \mathbf{a}_i \rangle = y_i$  and  $\langle p'_i, \mathbf{a}'_i \rangle = y'_i$ ; taking the same linear combination used to solve the equations above would give us  $p_i(0) + p_i(1) = e_{i-1}$  and  $p_i(r_i) = e_i$ . Thus, we have extracted valid polynomials for the information-theoretic sumcheck protocol.  $\square$

Putting together the above special soundness results for the subprotocols, we obtain special soundness for Spartan-NIZK.

**Lemma 5.6.** *Spartan-NIZK satisfies  $\mathbf{n}$ -computational special soundness, where*

$$\mathbf{n} = (1, (1, 2_{\text{li}}, 2)^\mu, 2, 2, 2, 1, (2, 2_{\text{li}}, 2)^\mu, \underbrace{(4_{\pm}, \dots, 4_{\pm})}_{\mu/2}, 2).$$

*Concretely, there exists a PPT tree extractor  $\mathcal{TE}_{\text{Spartan-NIZK}}$  such that for every EPT adversary  $\mathcal{A}$  against SS of Spartan-NIZK, there exists an EPT adversary  $\mathcal{B}$  against DL-REL, as efficient as  $\mathcal{A}$  and  $\mathcal{TE}_{\text{Spartan-NIZK}}$  combined, such that*

$$\text{Adv}_{\text{Spartan-NIZK}, \mathbf{n}}^{\text{SS}}(\mathcal{TE}_{\text{Spartan-NIZK}}, \mathcal{A}) \leq \text{Adv}_{\mathbb{G}, \sqrt{m}+2}^{\text{DL-REL}}(\mathcal{B}) + \frac{6\mu + 1}{|\mathbb{F}|}.$$

*Proof.* We describe the tree extractor  $\mathcal{TE}_{\text{Spartan-NIZK}}$ . Given a  $\mathbf{n}$ -tree of accepting transcripts, it runs the following sub-extractors for the corresponding sub-trees:

1. Run  $\mathcal{TE}_{\text{SC}}$  for the first sumcheck subprotocol on each  $(1, 2_{\text{li}}, 2)^\mu$  sub-tree to extract polynomials  $p_i(X)$  for  $i \in [\mu]$  that satisfy the information-theoretic sumcheck protocol.
2. Run  $\mathcal{TE}_{\text{ProdPf}}, \mathcal{TE}_{\text{OpenPf}}, \mathcal{TE}_{\text{EqPf}}$  on each corresponding 2-subtree to extract claims  $v_A, v_B, v_C$  such that  $e_x = (v_A \cdot v_B - v_C) \cdot \tilde{\text{eq}}(r_x, \tau)$ .
3. Run  $\mathcal{TE}_{\text{SC}}$  for the second sumcheck subprotocol on each  $(1, 2_{\text{li}}, 2)^\mu$  sub-tree to extract polynomials  $p_i(X)$  for  $i \in [\mu]$  that satisfy the information-theoretic sumcheck protocol.
4. Run  $\mathcal{TE}_{\text{PC}_{\text{Multi}}}$  for the opening argument  $\text{PC}_{\text{Multi}}.\text{Open}$  on the  $\underbrace{(4_{\pm}, \dots, 4_{\pm}, 2)}_{\mu/2}$  sub-tree, and on  $2^{\mu/2} = \sqrt{m}$  different challenges  $r_y$  provided by the  $(2, 2_{\text{li}}, 2)^\mu$  sub-tree, to extract a multilinear polynomial  $\tilde{w}(X)$  along with a correct evaluation  $\tilde{w}(r_y) = v_w$ .
5. Run  $\mathcal{TE}_{\text{EqPf}}$  for the final equality proof to verify the equality  $e_y = (r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C) \cdot v_Z$ .
6. Output the R1CS witness  $w$ .

Note that the  $(2, 2_{\text{li}}, 2)^\mu$  sub-tree in the second sumcheck subprotocol is necessary for extracting both from sumcheck, as well as from  $\text{PC}_{\text{Multi}}.\text{Open}$ . We now consider the following hybrids.  $\text{Hyb}_0$  corresponds to the game  $\text{SS}$  for  $\text{Spartan-NIZK}$  with the tree extractor constructed above.  $\text{Hyb}_1$  is the same as  $\text{Hyb}_0$ , but we additionally reject if the extracted R1CS witness is not satisfying. Conditioned on the event that none of the sub-extractor fails (and when that happens we get a DL-REL adversary  $\mathcal{B}$ ),  $\text{Hyb}_1$  differs from  $\text{Hyb}_0$  exactly when the soundness of  $\text{Spartan-Core}$  is violated, which happens with probability at most  $\frac{6\mu+1}{|\mathbb{F}|}$ .  $\square$

Using Lemma 4.6 with Lemma 5.6, we conclude that  $\text{Spartan-NIZK}_{\text{FS}}$  satisfies knowledge soundness. Note that the minimum partition size in the  $\mathbf{n}$ -tree of transcripts is  $C = \frac{|\mathbb{F}|-1}{2}$ .

**Theorem 5.7.** *Spartan-NIZK<sub>FS</sub> satisfies knowledge soundness. In particular, there exists an extractor  $\mathcal{E}_{\text{Spartan-NIZK}_{\text{FS}}}$  such that for every PPT prover  $\mathcal{P}^*$  against KS of Spartan-NIZK making at most  $Q$  random oracle queries, there exists an EPT adversary  $\mathcal{B}$  against DL-REL such that*

$$\text{Adv}_{\text{Spartan-NIZK}_{\text{FS}}}^{\text{KS}}(\mathcal{E}_{\text{Spartan-NIZK}_{\text{FS}}}, \mathcal{P}^*) \leq \frac{Q(Q-1) + (Q+1)(13\mu+10) + 2(6\mu+1)}{|\mathbb{F}|-1} + \text{Adv}_{\mathbb{G}, \sqrt{m}+2}^{\text{DL-REL}}(\mathcal{B}).$$

Here  $\mu = \log m$ . Both  $\mathcal{B}$  and the extractor  $\mathcal{E}_{\text{Spartan-NIZK}_{\text{FS}}}$  runs in expected time that is at most  $O(Q \cdot m^6)$  the running time of  $\mathcal{P}^*$ .

Our next task is to exhibit a  $k$ -ZK simulator for  $\text{Spartan-NIZK}_{\text{FS}}$ . The high-level idea is to let the simulator execute all subprotocols except the last with valid witnesses, then only invoke the simulator for the final  $\text{EqPf}$ .

**Theorem 5.8.** *Spartan-NIZK<sub>FS</sub> satisfies  $(r-1)$ -ZK, where  $r = 7\mu + 11$  is the number of rounds of Spartan-NIZK.*

*Proof.* See Fig. 8 for a pseudocode description of our simulators. Using the sum-check sub-simulator  $\mathcal{S}_{\text{SC}_{\text{FS}}}$  in the top of the figure, we build the full simulator  $\mathcal{S}_{\text{FS},r-1}$  for  $\text{Spartan-NIZK}_{\text{FS}}$ . From the construction of  $\mathcal{S}_{\text{FS},r-1}$ , it is clear that the proofs produced are accepting; this is because all the verifier's checks are done by checking the various proofs, which are either honestly generated, in which case validity follows from completeness, or by invoking the simulator, in which case validity follows from NIZK guarantee. Furthermore,  $\mathcal{S}_{\text{Spartan-NIZK}_{\text{FS},r-1}}$  only makes a *single* RO reprogramming, which when the simulator  $\mathcal{S}_{\text{EqPf}_{\text{FS}}}$  is invoked.

It remains to show that the output is indistinguishable from that of real transcripts. For the subprotocols, namely the  $\Sigma$ -protocols along with  $\text{PC}_{\text{Multi.Open}_{\text{FS}}}$ , that we generate transcripts by generating honest proofs, we argue that they are indistinguishable. Firstly, the inputs to the arguments are the same (being perfectly blinded commitment). Secondly, the sub-protocols themselves are zero-knowledge, which implies witness indistinguishability. This further implies that the honestly generated proofs made by our simulator are identically distributed as proofs in real transcripts. In the last sub-protocol  $\text{EqPf}_{\text{FS}}$  for which we use the simulator, we argue indistinguishability using the guarantee of the simulator  $\mathcal{S}_{\text{EqPf}_{\text{FS}}}$ . This concludes the proof of  $k$ -ZK.  $\square$

**Lemma 5.9.** *Spartan-NIZK<sub>FS</sub> satisfies perfect  $(r - 1)$ -UR.*

*Proof.* The last two rounds of  $\text{Spartan-NIZK}_{\text{FS}}$  consists of an instance of the  $\Sigma$ -protocol  $\text{EqPf}_{\text{FS}}$ , which itself satisfies perfect 1-UR. In more detail, the last message in  $\text{EqPf}_{\text{FS}}$  must be the unique scalar  $z$  that satisfies  $h^z = (C_1/C_2)^c \cdot \alpha$ , where  $C_1, C_2, \alpha$  are group elements determined by the previous messages. Hence  $\text{Spartan-NIZK}_{\text{FS}}$  satisfies perfect  $(r - 1)$ -UR.  $\square$

Combining our results above, we obtain SIM-EXT for  $\text{Spartan-NIZK}_{\text{FS}}$ .

**Theorem 5.10.** *Spartan-NIZK<sub>FS</sub> is simulation-extractable. Concretely, there exists a simulator-extractor  $\mathcal{E}_{\text{Spartan-NIZK}_{\text{FS}}}$  such that for every PPT adversary  $\mathcal{P}^*$  against SIM-EXT, there exists an EPT adversary  $\mathcal{B}$  against DL-REL such that*

$$\begin{aligned} \text{Adv}_{\text{Spartan-NIZK}_{\text{FS}}, \mathcal{R}_{\text{RICS}}}^{\text{SIM-EXT}}(\mathcal{S}_{\text{Spartan-NIZK}_{\text{FS},k}}, \mathcal{E}_{\text{Spartan-NIZK}_{\text{FS}}}, \mathcal{P}^*) \\ \leq \frac{Q(Q-1) + (Q+1)(13\mu+10) + 2(6\mu+1) + 2}{|\mathbb{F}| - 1} + \text{Adv}_{\mathcal{G}, \sqrt{m}+2}^{\text{DL-REL}}(\mathcal{B}). \end{aligned}$$

Both  $\mathcal{B}$  and  $\mathcal{E}_{\text{Spartan-NIZK}_{\text{FS}}}$  runs in expected time at most  $O(Q \cdot m^6)$  that of  $\mathcal{P}^*$ .

### 5.3 SIM-EXT of Spartan-SNARK

For  $\text{Spartan-SNARK}$ , the proof of SIM-EXT is similar to that of  $\text{Spartan-NIZK}$ . In particular, the proofs of  $k$ -ZK and  $k$ -UR carries over, and we only need to modify the proof of special soundness to accommodate for the more complex sparse multilinear polynomial commitment scheme. In what follows, we let  $r' = 7\mu + 11 + O(\mu)$  be the round complexity of  $\text{Spartan-SNARK}$ .

**Lemma 5.11.** *Spartan-SNARK<sub>FS</sub> satisfies  $k$ -ZK, where  $k = r' - 1$ .*

**Sub-simulator**  $\mathcal{S}_{\text{SCFS}}$ .

**Input.** Public parameters  $\text{pp}$ , a commitment  $C_e$  to some value  $e$  allegedly equal to  $\sum_{x \in \{0,1\}^\mu} p(x)$ , with  $p$  a multivariate polynomial of individual degree at most  $d$ , and previous transcript  $\text{tr}$  (including  $\text{pp}$ , R1CS input  $(A, B, C, \text{io})$ , and all prover's messages so far).

Set  $e_0 = e$ . For  $i = 1, \dots, \mu$ :

1. Sample  $p_i(X) \xleftarrow{\$} \mathbb{F}^{\leq d}[X]$  randomly conditioned on  $p_i(0) + p_i(1) = e_{i-1}$ . Compute a commitment  $C_{p_i} \leftarrow \text{Commit}(\text{pp}, p_i; \omega_{p_i})$  and append  $C_{p_i}$  to  $\text{tr}$ .
2. Obtain challenge  $r_i \leftarrow \text{H}(\text{tr})$ .
3. Let  $e_i = p_i(r_i)$ , compute a commitment  $C_{e_i} \leftarrow \text{Commit}(\text{pp}, e_i; \omega_{e_i})$ , and append  $C_{e_i}$  to  $\text{tr}$ .
4. Obtain challenges  $w_{i,1}, w_{i,2} \leftarrow \text{H}(\text{tr})$ .
5. Compute  $\mathbf{a}, C_{y_i}, y_i, \omega_{y_i}$  as specified in the sumcheck subprotocol. Generate an honest proof  $\pi_i \leftarrow \mathcal{P}_{\text{DotProdPfFS}}(\text{pp}, (C_{p_i}, C_{y_i}, \mathbf{a}_i), (p_i, \omega_{p_i}, y_i, \omega_{y_i}))$ . Append  $\pi_i$  to  $\text{tr}$ .

After  $\mu$  rounds, return  $C_{e_\mu}$ .

**Simulator**  $\mathcal{S}_{\text{FS}, r-1}(\text{pp}, (\mathbb{F}, A, B, C, \text{io}))$ :

Initialize  $\text{tr} = (\text{pp}, A, B, C, \text{io})$ .

1. Sample a random multilinear polynomial  $\tilde{w} \xleftarrow{\$} \mathbb{F}[X_1, \dots, X_\mu]$ . Compute  $C_{\tilde{w}} \leftarrow \text{PC}_{\text{Multi}}\text{-Commit}(\text{pp}, \tilde{w}; \omega_{\tilde{w}})$ , and append  $C_{\tilde{w}}$  to  $\text{tr}$ .
2. Obtain challenge  $\tau \leftarrow \text{H}(\text{tr})$ .
3. Run  $\mathcal{S}_{\text{SCFS}}$  on  $(\text{pp}, C_e)$  with current transcript  $\text{tr}$ , and get output  $C_{e_x} \leftarrow \text{Commit}(\text{pp}, e_x; \omega_{e_x})$  for some scalar  $e_x \in \mathbb{F}$ .
4. Sample  $v_A, v_B, v_C \xleftarrow{\$} \mathbb{F}$  at random conditioned on  $(v_A \cdot v_B - v_C) \cdot \tilde{e}\tilde{q}(r_x, \tau) = e_x$ , and set  $v_{AB} = v_A \cdot v_B$ . Compute  $C_{v_M} \leftarrow \text{Commit}(\text{pp}, v_M; \omega_M)$  for  $M \in \{A, B, C\}$  along with  $C_{v_{AB}} \leftarrow \text{Commit}(\text{pp}, v_{AB}; \omega_{AB})$ , and append them to  $\text{tr}$ .
5. Generate an honest proof

$$\pi_{\text{ProdPf}} \leftarrow \mathcal{P}_{\text{ProdPfFS}}(\text{pp}, (C_{v_A}, C_{v_B}, C_{v_{AB}}), (v_A, v_B, \omega_{v_A}, \omega_{v_B}, \omega_{v_{AB}})),$$

and append it to  $\text{tr}$ .

6. Generate an honest proof  $\pi_{\text{OpenPf}} \leftarrow \mathcal{P}_{\text{OpenPfFS}}(\text{pp}, (C_{v_C}), (v_C, \omega_{v_C}))$ , and append it to  $\text{tr}$ .
7. Generate an honest proof  $\pi_{\text{EqPf}, 1} \leftarrow \mathcal{P}_{\text{EqPfFS}}(\text{pp}, (C_{e_x}, C_{v'}), (\omega_{e_x} - \omega_{v'}))$ , where  $v' = (v_A \cdot v_B - v_C) \cdot \tilde{e}\tilde{q}(r_x, \tau)$  and  $C_{v'} = (C_{v_{AB}}/C_{v_C})^{\tilde{e}\tilde{q}(r_x, \tau)}$ ; then append it to  $\text{tr}$ .
8. Obtain challenges  $r_A, r_B, r_C \leftarrow \text{H}(\text{tr})$ .
9. Compute  $C_T = r_A \cdot C_{v_A} + r_B \cdot C_{v_B} + r_C \cdot C_{v_C}$ . Run  $\mathcal{S}_{\text{SCFS}}$  on  $(\text{pp}, C_T, \text{tr})$ , obtaining output  $C_{e_y} \leftarrow \text{Commit}(\text{pp}, e_y; \omega_{e_y})$ .
10. Generate opening proof  $\pi_{\text{PC}_{\text{Multi}}\text{-Open}} \leftarrow \mathcal{P}_{\text{PC}_{\text{Multi}}\text{-OpenFS}}(\text{pp}, (C_{\tilde{w}}, r_y), (\tilde{w}, \omega_{\tilde{w}}))$ ; at the end, get  $C_{v_w} = \text{Commit}(\text{pp}, v_w; \omega_{v_w})$ , where  $v_w \leftarrow \tilde{w}(r_y[1 \dots])$ , and append it to  $\text{tr}$ .
11. Compute  $v_Z = (1 - r_y[0]) \cdot v_w + r_y[0] \cdot (\tilde{\text{io}}, 1)(r_y[1 \dots])$  and  $C_{v_Z} = C_{v_w}^{1 - (r_y)_0} \cdot C_{v_{\text{io}}}^{(r_y)_0}$ .
12. Compute  $v_1 = \tilde{A}(r_x, r_y)$ ,  $v_2 = \tilde{B}(r_x, r_y)$ ,  $v_3 = \tilde{C}(r_x, r_y)$ . Generate a *simulated* proof  $\pi_{\text{EqPf}, 2} \leftarrow \mathcal{S}_{\text{EqPfFS}}$  for the equality  $e_y = (r_A \cdot v_1 + r_B \cdot v_2 + r_C \cdot v_3) \cdot v_Z$ . Append the proof to  $\text{tr}$ .

Return  $\text{tr}$ .

**Fig. 8.** Simulators for proof of  $k$ -ZK for Spartan.

*Proof.* We minimally modify the  $k$ -ZK simulator of  $\text{Spartan-NIZK}_{\mathbb{F}_S}$  to also output opening proofs  $\text{PC}_{\text{SparseMulti-Open}}_{\mathbb{F}_S}$  for  $\widetilde{M}(r_x, r_y)$  with  $M \in \{A, B, C\}$ . Since  $A, B, C$  are part of the public input, the simulator has full access to the matrices, and hence can produce the proofs honestly.  $\square$

**Lemma 5.12.** *Spartan-SNARK $_{\mathbb{F}_S}$  satisfies perfect  $k$ -UR, where  $k = r' - 1$ .*

*Proof.* Since  $\text{Spartan-SNARK}_{\mathbb{F}_S}$  ends with the same invocation of the equality proof  $\text{EqPf}_{\mathbb{F}_S}$ , we obtain the same result as Lemma 5.9.  $\square$

The proof of knowledge soundness for  $\text{Spartan-SNARK}_{\mathbb{F}_S}$  is similar to that of  $\text{Spartan-NIZK}_{\mathbb{F}_S}$ , except we further need to extract the polynomials involved in  $\text{PC}_{\text{SparseMulti-Open}}$ . We prove the lemma below in the full version.

**Lemma 5.13.** *Spartan-SNARK $_{\mathbb{F}_S}$  satisfies knowledge soundness. Concretely, there exists an extractor  $\mathcal{E}_{\text{Spartan-SNARK}_{\mathbb{F}_S}}$  such that for every PPT prover  $\mathcal{P}^*$  against KS of  $\text{Spartan-SNARK}_{\mathbb{F}_S}$  making at most  $Q$  random oracle queries, there exists an EPT adversary  $\mathcal{B}$  against DL-REL such that*

$$\begin{aligned} & \text{Adv}_{\text{Spartan-SNARK}_{\mathbb{F}_S}}^{\text{KS}}(\mathcal{E}_{\text{Spartan-SNARK}_{\mathbb{F}_S}}, \mathcal{P}^*) \\ & \leq \frac{Q(Q-1) + (Q+1)(25\mu + 9\nu + 16) + 6(m+n) + O(\mu + \nu)}{|\mathbb{F}| - 1} + \text{Adv}_{\mathbb{G}, \sqrt{m+n+2}}^{\text{DL-REL}}(\mathcal{B}). \end{aligned}$$

Here  $\mu = \log m, \nu = \log n$ . Both  $\mathcal{B}$  and the extractor  $\mathcal{E}_{\text{Spartan-SNARK}_{\mathbb{F}_S}}$  runs in expected time that is at most  $O(Q \cdot m^{7.5} \cdot (m+n)^3)$  the running time of  $\mathcal{P}^*$ .

Combining the results above, we obtain SIM-EXT for  $\text{Spartan-SNARK}_{\mathbb{F}_S}$ .

**Theorem 5.14.** *Spartan-SNARK $_{\mathbb{F}_S}$  satisfies SIM-EXT. Concretely, there exists a simulator-extractor  $\mathcal{E}_{\text{Spartan-SNARK}_{\mathbb{F}_S}}$  such that for every PPT adversary  $\mathcal{P}^*$  against SIM-EXT of  $\text{Spartan-SNARK}_{\mathbb{F}_S}$ , there exists an EPT adversary  $\mathcal{B}$  against DL-REL with*

$$\begin{aligned} & \text{Adv}_{\text{Spartan-SNARK}_{\mathbb{F}_S}, \mathcal{R}_{\text{R1CS}}}^{\text{SIM-EXT}}(\mathcal{S}_{\text{Spartan-SNARK}_{\mathbb{F}_S, k}}, \mathcal{E}_{\text{Spartan-SNARK}_{\mathbb{F}_S}}, \mathcal{P}^*) \\ & \leq \frac{Q(Q-1) + (Q+1)(25\mu + 9\nu + 16) + 6(m+n) + O(\mu + \nu)}{|\mathbb{F}| - 1} + \text{Adv}_{\mathbb{G}, \sqrt{m+n+2}}^{\text{DL-REL}}(\mathcal{B}). \end{aligned}$$

$\mathcal{B}$  and  $\mathcal{E}_{\text{Spartan-SNARK}_{\mathbb{F}_S}}$  run in expected time  $O(Q \cdot m^{7.5} \cdot (m+n)^3)$  that of  $\mathcal{P}^*$ .

## 6 Simulation Extractability of Bulletproofs

In this section, we show that the Bulletproofs protocols in [16] satisfy SIM-EXT, without relying on the AGM. The authors of [16] introduced two protocols, an *aggregate range proof* BP-ARP and an *arithmetic circuit satisfiability proof* BP-ACSPf<sup>2</sup>, with both building on an *inner product argument* BP-IPA.

<sup>2</sup> To keep the naming consistent with [16], we refer to them as *proofs* even though they are actually *arguments*.

### 6.1 Aggregate Range Proof

We give a full description of the aggregate range proof BP-ARP in Fig. 9. The value  $m$  is the number of committed values  $v_i$ , and  $n$  is the bit length of the upper bound (i.e., we prove  $v_i \in [0, 2^n - 1]$  for all  $i \in [m]$ ). Following the same approach as in Sect. 5 for Spartan, we need to establish three properties of BP-ARP<sub>FS</sub>: (1) knowledge soundness, (2) the existence of a  $k$ -ZK simulator, and (3)  $k$ -UR for the same round  $k$ . We begin with the proof of knowledge soundness, which is essentially a restatement of the original result from [16].

**Lemma 6.1.** BP-ARP<sub>FS</sub> satisfies  $(m \cdot n, m + 2, 3, 2, \underbrace{4_{\pm}, \dots, 4_{\pm}}_{\log(m \cdot n)})$ -computational special soundness, and hence knowledge soundness. Concretely, there exists an extractor  $\mathcal{E}_{\text{BP-ARP}_{\text{FS}}}$  such that for every PPT adversary  $\mathcal{P}^*$  against KS making at most  $Q$  random oracle queries, there exists an adversary  $\mathcal{A}$  against DL-REL with

$$\text{Adv}_{\text{BP-ARP}_{\text{FS}}}^{\text{KS}}(\mathcal{E}_{\text{BP-ARP}_{\text{FS}}}, \mathcal{P}^*) \leq \text{Adv}_{\mathbb{G}, 2mn+3}^{\text{DL-REL}}(\mathcal{A}) + \frac{Q(Q-1) + 2(Q+1)(m(n+1) + 3 \log(m \cdot n) + 3)}{|\mathbb{F}| - 1}.$$

Both  $\mathcal{A}$  and the extractor  $\mathcal{E}_{\text{BP-ARP}_{\text{FS}}}$  run in expected time that is at most  $O(Q \cdot m^4 \cdot n^3)$  times the runtime of  $\mathcal{P}^*$ .

*Proof.* The description of a tree extractor  $\mathcal{T}\mathcal{E}_{\text{BP-ARP}}$ , which either outputs a valid witness or a discrete log relation, can be found in [16]. This concludes the proof of computational special soundness. Combining Theorem 4.4 with Lemma 4.6, we conclude knowledge soundness for BP-ARP<sub>FS</sub>. The expected runtime of the extractor  $\mathcal{E}_{\text{BP-ARP}_{\text{FS}}}$ , as well as the adversary  $\mathcal{A}$ , is at most  $O(Q \cdot (mn) \cdot (m + 2) \cdot 6 \cdot (mn)^2) = O(Q \cdot m^4 \cdot n^3)$  times the runtime of  $\mathcal{P}^*$ , by Theorem 4.4.  $\square$

**Lemma 6.2.** BP-ARP<sub>FS</sub> satisfies perfect 2-ZK.

*Proof.* We present the 2-ZK simulator  $\mathcal{S}_{\text{BP-ARP}_{\text{FS}},x}$  in Fig. 10, and argue that its output is identically distributed to the output of an honest prover.

All the challenges are chosen randomly as with real proofs. Next, in both real and simulated proofs, the proof elements  $A, T_1, \beta_x, \mu$  and the underlying vectors  $\mathbf{l}, \mathbf{r}$  are distributed uniformly among their respective domains. The proof elements  $S, T_2$  are then uniquely determined from the previous ones from the verification equations that they must satisfy. Finally, both the scalar  $\hat{t}$  and the inner product argument  $\pi_{\text{BP-IPA}}$  is generated deterministically from  $\mathbf{l}, \mathbf{r}$ ; this implies identical distributions for those proof elements as well.  $\square$

Finally, we show the 2-UR property of BP-ARP. This result relies on the fact that BP-IPA<sub>FS</sub> has computationally unique proofs, shown in the full version.

**Lemma 6.3.** BP-ARP<sub>FS</sub> satisfies 2-UR. In particular, for any adversary  $\mathcal{A}$  against 2-UR of BP-ARP<sub>FS</sub>, there exists an adversary  $\mathcal{B}$  against DL-REL such that

$$\text{Adv}_{\text{BP-ARP}_{\text{FS}}}^{2\text{-UR}}(\mathcal{A}) \leq 2 \cdot \frac{Q(Q-1) + 6(Q+1) \log mn}{|\mathbb{F}| - 1} + 3 \cdot \text{Adv}_{\mathbb{G}, 2mn+3}^{\text{DL-REL}}(\mathcal{B}).$$



**Aggregate Range Proof Relation.**

$$\mathcal{R}_{\text{BP-ARP}} = \left\{ \begin{array}{l} ((m, n, \mathbf{g}, \mathbf{h}, g, h, u), \mathbf{V}, (\mathbf{v}, \boldsymbol{\gamma})) : \\ V_j = g^{v_j} h^{\gamma_j} \wedge v_j \in [0, 2^n - 1] \forall j \in [1, m] \end{array} \right\}.$$

**Interaction Phase.** Denote  $\mathbf{y}^{m \cdot n} = (1, y, \dots, y^{m \cdot n - 1}) \in \mathbb{F}^{m \cdot n}$ .

1.  $\mathcal{P}$  samples  $\alpha, \rho \xleftarrow{\$} \mathbb{F}$ ,  $\mathbf{s}_L, \mathbf{s}_R \xleftarrow{\$} \mathbb{F}^{m \cdot n}$  and computes

$$\begin{aligned} \mathbf{a}_L &\in \{0, 1\}^{m \cdot n} \text{ such that } \langle (\mathbf{a}_L)_{[(j-1)n, jn-1]}, \mathbf{2}^n \rangle = v_j \forall j \in [1, m], \\ \mathbf{a}_R &= \mathbf{a}_L - \mathbf{1}^{m \cdot n}, \\ A &= h^\alpha \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R}, \quad S = h^\rho \mathbf{g}^{\mathbf{s}_L} \mathbf{h}^{\mathbf{s}_R}. \end{aligned}$$

$\mathcal{P}$  sends  $A, S$  to  $\mathcal{V}$ .

2.  $\mathcal{V}$  sends challenges  $y, z \xleftarrow{\$} \mathbb{F}^*$ .

3.  $\mathcal{P}$  samples  $\beta_1, \beta_2 \xleftarrow{\$} \mathbb{F}$  and computes

$$\begin{aligned} \ell(X) &= (\mathbf{a}_L - z \cdot \mathbf{1}^{m \cdot n}) + \mathbf{s}_L \cdot X, \\ r(X) &= \mathbf{y}^{m \cdot n} \circ (\mathbf{a}_R + z \cdot \mathbf{1}^{m \cdot n} + \mathbf{s}_R \cdot X) + \sum_{j=1}^m z^{j+1} \cdot \left( \mathbf{0}^{(j-1)n} \|\mathbf{2}^n\| \mathbf{0}^{(m-j)n} \right), \\ t(X) &= \langle \ell(X), r(X) \rangle = t_0 + t_1 \cdot X + t_2 \cdot X^2, \quad T_1 = g^{t_1} h^{\beta_1}, \quad T_2 = g^{t_2} h^{\beta_2}. \end{aligned}$$

$\mathcal{P}$  sends  $T_1, T_2$  to  $\mathcal{V}$ .

4.  $\mathcal{V}$  sends challenge  $x \xleftarrow{\$} \mathbb{F}^*$ .

5.  $\mathcal{P}$  computes

$$\begin{aligned} \mathbf{l} &= \ell(x), \quad \mathbf{r} = r(x), \quad \hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle, \quad \mu = \alpha + \rho \cdot x, \\ \beta_x &= \beta_2 \cdot x^2 + \beta_1 \cdot x + \sum_{j=1}^m z^{j+1} \cdot \gamma_j. \end{aligned}$$

$\mathcal{P}$  sends  $\hat{t}, \beta_x, \mu$  to  $\mathcal{V}$ .

6.  $\mathcal{V}$  sends challenge  $w \xleftarrow{\$} \mathbb{F}^*$ .

7.  $\mathcal{P}, \mathcal{V}$  both compute

$$\begin{aligned} \mathbf{h}' &= \mathbf{h}^{\mathbf{y}^{m \cdot n}}, \quad u' = u^w, \\ P' &= h^{-\mu} \cdot A \cdot S^x \cdot \mathbf{g}^{-z \cdot \mathbf{1}^{m \cdot n}} \cdot (\mathbf{h}')^{z \cdot \mathbf{y}^{m \cdot n}} \cdot \prod_{j=1}^m (\mathbf{h}')_{[(j-1)n, jn-1]}^{z^{j+1} \cdot \mathbf{2}^n} \cdot (u')^{\hat{t}}. \end{aligned}$$

8.  $\mathcal{P}, \mathcal{V}$  engage in BP-IPA for the triple  $((m \cdot n, \mathbf{g}, \mathbf{h}', u'), P', (\mathbf{l}, \mathbf{r}))$ .

**Verification.**

1.  $\mathcal{V}$  rejects if BP-IPA fails.

2.  $\mathcal{V}$  computes  $R = \mathbf{V}^{z^2 \cdot \mathbf{z}^m} \cdot g^{(z-z^2) \cdot \langle \mathbf{1}^{m \cdot n}, \mathbf{y}^{m \cdot n} \rangle - \sum_{j=1}^m z^{j+2} \cdot \langle \mathbf{1}^n, \mathbf{2}^n \rangle} \cdot T_1^x \cdot T_2^{x^2}$ .

3.  $\mathcal{V}$  checks whether  $g^{\hat{t}} h^{\beta_x} \stackrel{?}{=} R$ .

**Fig. 9.** Bulletproofs' Aggregate Range Proof BP-ARP

**Simulator**  $\mathcal{S}_{\text{BP-ARPF}_S, x}(\text{pp} = (m, n, \mathbf{g}, \mathbf{h}, g, h, u), \mathbf{V})$ :

1. Initialize  $\text{tr} = (\text{pp}, \mathbf{V})$ . Sample  $\alpha, \rho \xleftarrow{\$} \mathbb{F}$ ,  $\mathbf{a}_L, \mathbf{a}_R, \mathbf{s}_L, \mathbf{s}_R \xleftarrow{\$} \mathbb{F}^{m \cdot n}$  and compute  $A = h^\alpha \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R}$ ,  $S = h^\rho \mathbf{g}^{\mathbf{s}_L} \mathbf{h}^{\mathbf{s}_R}$ . Append  $A, S$  to  $\text{tr}$ .
2. Obtain challenges  $y, z \leftarrow \text{H}(\text{tr})$ .
3. Sample  $x \xleftarrow{\$} \mathbb{F}^*$  and compute  $\mu = \alpha + \rho \cdot x$ ,  $\mathbf{l} = (\mathbf{a}_L - z \cdot \mathbf{1}^{m \cdot n}) + \mathbf{s}_L \cdot x$ ,  $\mathbf{r} = \mathbf{y}^{m \cdot n} \circ (\mathbf{a}_R + z \cdot \mathbf{1}^{m \cdot n} + \mathbf{s}_R \cdot x) + \sum_{j=1}^m z^{j+1} \cdot (\mathbf{0}^{(j-1)n} \parallel \mathbf{2}^n \parallel \mathbf{0}^{(m-j)n})$ , and  $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ .
4. Sample  $\beta_x \xleftarrow{\$} \mathbb{F}$ ,  $T_1 \xleftarrow{\$} \mathbb{G}$ , and compute  $T_2 = \left( g^{\hat{t} - \delta(y, z)} \cdot h^{\beta_x} \cdot \mathbf{V}^{-z^2 \cdot \mathbf{z}^m} \cdot T_1^{-x} \right)^{x^{-2}}$ , where  $\delta(y, z) = (z - z^2) \cdot \langle \mathbf{1}^{m \cdot n}, \mathbf{y}^{m \cdot n} \rangle - \sum_{j=1}^m z^{j+2} \cdot \langle \mathbf{1}^n, \mathbf{z}^n \rangle$ . Append  $T_1, T_2$  to  $\text{tr}$ .
5. Reprogram  $\text{H}(\text{tr}) := x$ , then append  $\hat{t}, \beta_x, \mu$  to  $\text{tr}$ .
6. Obtain challenge  $w \leftarrow \text{H}(\text{tr})$ .
7. Compute  $\mathbf{h}' = \mathbf{h}^{\mathbf{y}^{-m \cdot n}}$ ,  $u' = u^w$ , and
 
$$P' = h^{-\mu} \cdot A \cdot S^x \cdot \mathbf{g}^{-z \cdot \mathbf{1}^{m \cdot n}} \cdot (\mathbf{h}')^{z \cdot \mathbf{y}^{m \cdot n}} \cdot \prod_{j=1}^m (\mathbf{h}')_{[(j-1)n, jn-1]}^{z^{j+1} \cdot \mathbf{2}^n} \cdot (u')^{\hat{t}}.$$
8. Generate an honest proof  $\pi_{\text{BP-IPAF}_S} \leftarrow \mathcal{P}_{\text{BP-IPAF}_S}((m \cdot n, \mathbf{g}, \mathbf{h}', u'), P', (\mathbf{l}, \mathbf{r}))$ .
9. Output  $\pi_{\text{BP-ARPF}_S} = (A, S, T_1, T_2, \hat{t}, \beta_x, \mu, \pi_{\text{BP-IPAF}_S})$ .

**Fig. 10.** BP-ARPF<sub>S</sub> *k*-ZK simulator

$\mathcal{B}$  runs in expected time at most  $O(Q \cdot m^2 \cdot n^2)$  that of  $\mathcal{A}$ 's runtime.

*Proof.* We proceed through a sequence of hybrids. The high-level idea is to analyze different cases for where the two proofs  $\pi, \pi'$  first differ after the  $x$  challenge, and reduce each case to breaking DL-REL or the unique proof property of BP-IPA (which in turn reduces to breaking DL-REL).

- $\text{Hyb}_0$  is the game  $2\text{-UR}_{\text{BP-ARPF}_S}^A$ . Recall that in this game, an adversary  $\mathcal{A}$  outputs an input  $\mathbf{V}$ , a challenge  $x \in \mathbb{F}^*$ , and two proofs  $\pi, \pi'$  that agrees up to the  $x$  challenge, i.e. we have  $\pi = (A, S, T_1, T_2, \hat{t}, \beta_x, \mu, \pi_{\text{BP-IPAF}_S})$  and  $\pi' = (A, S, T_1, T_2, \hat{t}', \beta'_x, \mu', \pi'_{\text{BP-IPAF}_S})$ .  $\mathcal{A}$  wins if  $\pi \neq \pi'$  and both proofs are accepting with respect to the  $x$  challenge that it chose.
- $\text{Hyb}_1$  is the same as  $\text{Hyb}_0$ , except that we also run  $\mathcal{E}_{\text{BP-IPAF}_S}$  on the proofs  $\pi_{\text{BP-IPAF}_S}, \pi'_{\text{BP-IPAF}_S}$  to extract witnesses  $(\mathbf{l}, \mathbf{r})$  and  $(\mathbf{l}', \mathbf{r}')$ .  $\text{Hyb}_1$  returns 0 if the extractor aborts on either proofs, or  $\hat{t} \neq \langle \mathbf{l}, \mathbf{r} \rangle$  or  $\hat{t}' \neq \langle \mathbf{l}', \mathbf{r}' \rangle$ .

We can see that  $\text{Hyb}_1$  is identical to  $\text{Hyb}_0$ , except when the extractor  $\mathcal{E}_{\text{BP-IPAF}_S}$  fails in extracting from either proofs  $\pi_{\text{BP-IPAF}_S}, \pi'_{\text{BP-IPAF}_S}$ . The probability that this happens is precisely bounded by twice the KS advantage of BP-IPAF<sub>S</sub>.

Concretely, invoking the extractor for BP-IPAF<sub>S</sub>, there exists an adversary  $\mathcal{B}$  against DL-REL, running in expected time at most  $O(Q \cdot m^2 \cdot n^2)$  that of  $\mathcal{A}$ 's runtime, such that

$$|\Pr[\text{Hyb}_0] - \Pr[\text{Hyb}_1]| \leq 2 \frac{Q(Q-1) + 6(Q+1) \log(m \cdot n)}{|\mathbb{F}| - 1} + 2 \text{Adv}_{\mathbb{G}, 2mn+3}^{\text{DL-REL}}(\mathcal{B}).$$

It remains to show that if  $\text{Hyb}_1$  returns 1, then there exists an adversary  $\mathcal{B}'$  that returns a non-trivial discrete log relation. Adversary  $\mathcal{B}'$  is as follows:

- **If  $\hat{t} \neq \hat{t}'$  or  $\beta_x \neq \beta'_x$ :** since both proofs are accepting and are the same up to the  $x$  challenge, we have

$$g^{\hat{t}} \cdot h^{\beta_x} = V^{z^2} \cdot g^{\delta(y,z)} \cdot T_1^x \cdot T_2^{x^2} = g^{\hat{t}'} \cdot h^{\beta'_x}.$$

- **If  $(\hat{t}, \beta_x) = (\hat{t}', \beta'_x)$  but  $\mu \neq \mu'$ :** since both proofs  $\pi_{\text{BP-IPAFS}}, \pi'_{\text{BP-IPAFS}}$  are accepting, we have

$$\begin{aligned} \mathbf{g}^1 \cdot \mathbf{h}^{(y^{-m \cdot n} \text{or})} \cdot h^\mu &= A \cdot S^x \cdot \mathbf{g}^{-z \cdot 1^{m \cdot n}} \cdot (\mathbf{h}')^{z \cdot \mathbf{y}^{m \cdot n}} \cdot \prod_{j=1}^m (\mathbf{h}')_{[(j-1)n, jn-1]}^{z^{j+1} \cdot 2^n} \cdot u^{w \cdot \hat{t}} \\ &= \mathbf{g}^{1'} \cdot \mathbf{h}^{(y^{-m \cdot n} \text{or}')} \cdot h^{\mu'}. \end{aligned}$$

- **If  $(\hat{t}, \beta_x, \mu) = (\hat{t}', \beta'_x, \mu')$  but  $\pi_{\text{BP-IPAFS}} \neq \pi'_{\text{BP-IPAFS}}$ :** here, we know that both BP-IPAFS proofs are for the same statement  $P'$ , with extracted witnesses  $(\mathbf{1}, \mathbf{r}), (\mathbf{1}', \mathbf{r}')$ . From the proof that BP-IPAFS is computationally unique, the two witnesses must be different, which gives a discrete log relation.

Note that the first two cases above give discrete log relations, and if  $\text{Hyb}_1$  returns 1, then  $\pi \neq \pi'$ , hence at least one of the above cases happens. Putting everything together and unifying  $\mathcal{B}, \mathcal{B}'$  we get the desired bound.  $\square$

We finally obtain SIM-EXT from the previous results and Theorem 3.4.

**Theorem 6.4.** *BP-ARPF<sub>FS</sub> satisfies SIM-EXT. In particular, there exists a simulator-extractor  $\mathcal{E}_{\text{BP-ARPF}_{\text{FS}}}$  such that for any adversary  $\mathcal{P}^*$  against SIM-EXT of BP-ARPF<sub>FS</sub>, there exists an adversary  $\mathcal{B}$  against DL-REL such that*

$$\begin{aligned} \text{Adv}_{\text{BP-ARPF}_{\text{FS}}}^{\text{SIM-EXT}}(\mathcal{E}_{\text{BP-ARPF}_{\text{FS}}}, \mathcal{P}^*) &\leq 4 \cdot \text{Adv}_{\mathbb{G}, 2mn+3}^{\text{DL-REL}}(\mathcal{B}) \\ &+ \frac{3Q(Q-1) + 2(Q+1)(m(n+1) + 6 \log(mn) + 3) + 2}{|\mathbb{F}| - 1}. \end{aligned}$$

$\mathcal{B}$  runs in expected time at most  $O(Q \cdot m^4 \cdot n^3)$  the runtime of  $\mathcal{P}^*$ .

## 6.2 Arithmetic Circuit Satisfiability Proof

We will describe BP-ACSPf and prove the following theorem in the full version.

**Theorem 6.5.** *BP-ACSPf<sub>FS</sub> satisfies SIM-EXT. Concretely, there exists a simulator-extractor  $\mathcal{E}_{\text{BP-ACSPf}_{\text{FS}}}$  such that for any adversary  $\mathcal{P}^*$  against SIM-EXT of BP-ACSPf<sub>FS</sub>, there exists an adversary  $\mathcal{B}$  against DL-REL such that*

$$\begin{aligned} \text{Adv}_{\text{BP-ACSPf}_{\text{FS}}}^{\text{SIM-EXT}}(\mathcal{E}_{\text{BP-ACSPf}_{\text{FS}}}, \mathcal{P}^*) &\leq 4 \cdot \text{Adv}_{\mathbb{G}, 2n+1}^{\text{DL-REL}}(\mathcal{B}) \\ &+ \frac{3Q(Q-1) + 2(Q+1)(n+q + 9 \log n + 6) + 2}{|\mathbb{F}| - 1}. \end{aligned}$$

Here  $n$  is the number of multiplication gates, and  $q$  is the number of committed inputs.  $\mathcal{B}$  runs in expected time at most  $O(Q \cdot q \cdot n^3)$  the runtime of  $\mathcal{P}^*$ .

## 7 Quantitative Discussion of Our SIM-EXT Bounds

In this section, we briefly show how to interpret the tightness of our SIM-EXT bounds for Bulletproofs and Spartan, and compare them with the previous analyses of [30,33] using AGM. We leave a detailed discussion to the full version.

By Theorem 3.4, we see that the SIM-EXT advantage is tightly related to the KS advantage. Thus, we will compare the latter. For BP-ARP with  $m = 1$  (range proof of a single value), we compare our KS bound with the AGM-based bound of [33] in Fig. 11. Our approach loses tightness due to two factors: first, we lose a factor of  $Q$  due to rewinding (shown to be somewhat inherent for the similar case of Schnorr signatures [26]), and second, our DL-REL adversary is *expected time*, which leads to another “square-root” loss in security [42] (namely  $\text{Adv}_{\mathbb{G},2n+3}^{\text{DL-REL}}(\mathcal{A}) \leq \sqrt{t(\mathcal{A})^2/|\mathbb{F}|}$  for generic attacks). Our concrete KS advantages for Spartan is even lower, due to the bigger tree sizes of Spartan. We leave achieving tighter rewinding-based bounds to future work.

	Lemma 6.1	[33, Theorem 4]
<i>Asymptotic</i>	$O\left(\frac{Q^2+Qn}{ \mathbb{F} }\right) + \text{Adv}_{\mathbb{G},2n+3}^{\text{DL-REL}}(\mathcal{A})$ where $\mathbb{E}[t(\mathcal{A})] = O(Q \cdot n^3 \cdot t(\mathcal{P}^*))$	$O\left(\frac{Qn}{ \mathbb{F} }\right) + \text{Adv}_{\mathbb{G},2n+3}^{\text{DL-REL}}(\mathcal{A}')$ where $t(\mathcal{A}') = O(Q \cdot n)$
<i>Concrete</i>	$\approx 22$ bits of security	$\approx 164$ bits of security

**Fig. 11.** Comparison of KS advantages, obtained by rewinding (ours) versus AGM [33], for Bulletproofs’ single range proof, e.g. BP-ARP with  $m = 1$ . Here  $t(\cdot)$  denotes the running time. For concrete advantage, we take  $|\mathbb{F}| \approx 2^{256}$ ,  $n = 64$ ,  $t(\mathcal{P}^*) = 2^{48}$ ,  $Q = 2^{40}$ .

## References

1. Aleo. <https://www.aleo.org/> (2022)
2. Attema, T., Fehr, S., Kloof, M.: Fiat-shamir transformation of multi-round interactive proofs. Cryptology ePrint Archive, Report 2021/1377 (2021). <https://eprint.iacr.org/2021/1377>
3. Baghery, K., Kohlweiss, M., Siim, J., Volkhov, M.: Another look at extraction and randomization of groth’s zk-SNARK. Cryptology ePrint Archive, Report 2020/811 (2020), <https://eprint.iacr.org/2020/811>
4. Baghery, K., Pindado, Z., Ràfols, C.: Simulation extractable versions of groth’s zk-SNARK revisited. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 2020. LNCS, vol. 12579, pp. 453–461. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-65411-5\\_22](https://doi.org/10.1007/978-3-030-65411-5_22)
5. Baghery, K., Sedaghat, M.: Tiramisu: black-box simulation extractable NIZKs in the updatable CRS model. Cryptology ePrint Archive, Report 2020/474 (2020). <https://eprint.iacr.org/2020/474>
6. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (May / Jun (2006)

7. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive*, Report 2018/046 (2018). <https://eprint.iacr.org/2018/046>
8. 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)
9. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17653-2\\_4](https://doi.org/10.1007/978-3-030-17653-2_4)
10. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53644-5\\_2](https://doi.org/10.1007/978-3-662-53644-5_2)
11. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: pitfalls of the fiat-shamir heuristic and applications to helios. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 626–643. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-34961-4\\_38](https://doi.org/10.1007/978-3-642-34961-4_38)
12. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKs and proof-carrying data. *Cryptology ePrint Archive*, Report 2012/095 (2012). <https://eprint.iacr.org/2012/095>
13. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo Infinite: proof-carrying data from additive polynomial commitments. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 649–680. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84242-0\\_23](https://doi.org/10.1007/978-3-030-84242-0_23)
14. 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](https://doi.org/10.1007/978-3-662-49896-5_12)
15. Bowe, S., Grigg, J., Hopwood, D.: Halo: recursive proof composition without a trusted setup. *Cryptology ePrint Archive*, Report 2019/1021 (2019). <https://eprint.iacr.org/2019/1021>
16. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press (May 2018)
17. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 677–706. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45721-1\\_24](https://doi.org/10.1007/978-3-030-45721-1_24)
18. Canetti, R., Sarkar, P., Wang, X.: Triply adaptive UC NIZK. *Cryptology ePrint Archive*, Report 2020/1212 (2020). <https://eprint.iacr.org/2020/1212>
19. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 78–96. Springer, Heidelberg (2006). [https://doi.org/10.1007/11818175\\_5](https://doi.org/10.1007/11818175_5)
20. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 738–768. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-45721-1\\_26](https://doi.org/10.1007/978-3-030-45721-1_26)
21. Coda. <https://codaprotocol.com/> (2022)
22. Decker, C., Wattenhofer, R.: Bitcoin transaction malleability and MtGox. In: Kutylowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8713, pp. 313–326. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-11212-1\\_18](https://doi.org/10.1007/978-3-319-11212-1_18)

23. Don, J., Fehr, S., Majenz, C.: The measure-and-reprogram technique 2.0: multi-round fiat-shamir and more. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 602–631. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-56877-1\\_21](https://doi.org/10.1007/978-3-030-56877-1_21)
24. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the fiat-shamir transform. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-34931-7\\_5](https://doi.org/10.1007/978-3-642-34931-7_5)
25. 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](https://doi.org/10.1007/3-540-47721-7_12)
26. Fleischhacker, N., Jager, T., Schröder, D.: On tight security proofs for Schnorr signatures. *J. Crypt.* **32**(2), 566–599 (2019)
27. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 33–62. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-96881-0\\_2](https://doi.org/10.1007/978-3-319-96881-0_2)
28. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive, Report 2019/953* (2019), <https://eprint.iacr.org/2019/953>
29. Ganesh, C., Khoshakhlagh, H., Kohlweiss, M., Nitulescu, A., Zając, M.: What makes fiat-shamir zkSNARKs (updatable SRS) simulation extractable? In: SCN (2022)
30. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 397–426. Springer, Heidelberg (May / Jun (2022)
31. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-shamir bulletproofs are non-malleable (in the random oracle model). *Cryptology ePrint Archive* (2023)
32. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-38348-9\\_37](https://doi.org/10.1007/978-3-642-38348-9_37)
33. Ghoshal, A., Tessaro, S.: Tight state-restoration soundness in the algebraic group model. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12827, pp. 64–93. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-84252-9\\_3](https://doi.org/10.1007/978-3-030-84252-9_3)
34. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: 17th ACM STOC, pp. 291–304. ACM Press (May 1985)
35. Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R.S.: Brakedown: linear-time and post-quantum SNARKs for R1CS. *Cryptology ePrint Archive, Report 2021/1043* (2021). <https://eprint.iacr.org/2021/1043>
36. Grin: a minimal implementation of mumblewimble. <https://github.com/mumblewimble/grin> (2022)
37. Groth, J.: Simulation-Sound NIZK proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 444–459. Springer, Heidelberg (2006). [https://doi.org/10.1007/11935230\\_29](https://doi.org/10.1007/11935230_29)
38. 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](https://doi.org/10.1007/978-3-662-49896-5_11)

39. 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](https://doi.org/10.1007/978-3-319-63715-0_20)
40. Grubbs, P., Arun, A., Zhang, Y., Bonneau, J., Walfish, M.: Zero-knowledge middleboxes. In: Butler, K.R.B., Thomas, K. (eds.) USENIX Security 2022, pp. 4255–4272. USENIX Association (2022)
41. Haines, T., Lewis, S.J., Pereira, O., Teague, V.: How not to prove your election outcome. In: 2020 IEEE Symposium on Security and Privacy, pp. 644–660. IEEE Computer Society Press (May 2020)
42. Jaeger, J., Tessaro, S.: Expected-time cryptography: generic techniques and applications to concrete soundness. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12552, pp. 414–443. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-64381-2\\_15](https://doi.org/10.1007/978-3-030-64381-2_15)
43. Jain, A., Pandey, O.: Non-malleable zero knowledge: black-box constructions and definitional relationships. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 435–454. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-10879-7\\_25](https://doi.org/10.1007/978-3-319-10879-7_25)
44. Kothapalli, A., Setty, S., Tzialla, I.: Nova: recursive zero-knowledge arguments from folding schemes. Cryptology ePrint Archive, Report 2021/370 (2021). <https://eprint.iacr.org/2021/370>
45. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2111–2128. ACM Press (Nov 2019)
46. Bulletproofs+ in monero. <https://www.getmonero.org/2020/12/24/Bulletproofs+-in-Monero.html> (2020)
47. Mt.Gox press release. <https://web.archive.org/web/20140214041924/>, [https://www.mtgox.com/press\\_release\\_20140210.html](https://www.mtgox.com/press_release_20140210.html) (2014)
48. 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)
49. Pass, R.: On deniability in the common reference string and random oracle model. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 316–337. Springer, Heidelberg (2003). [https://doi.org/10.1007/978-3-540-45146-4\\_19](https://doi.org/10.1007/978-3-540-45146-4_19)
50. Pass, R., Rosen, A.: New and improved constructions of non-malleable cryptographic protocols. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC, pp. 533–542. ACM Press (May 2005)
51. Polygon. <https://polygon.technology/> (2022)
52. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th FOCS, pp. 543–553. IEEE Computer Society Press (Oct 1999)
53. Scroll. <https://scroll.io/> (2022)
54. Setty, S.: Spartan: efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 704–737. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-56877-1\\_25](https://doi.org/10.1007/978-3-030-56877-1_25)
55. Setty, S., Lee, J.: Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275 (2020). <https://eprint.iacr.org/2020/1275>
56. Starkware. <https://starkware.co/> (2022)

57. Wahby, R.S., Tzialla, I., shelat, a., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy, pp. 926–943. IEEE Computer Society Press (May 2018)
58. Wee, H.: Zero knowledge in the random Oracle model, revisited. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 417–434. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-10366-7\\_25](https://doi.org/10.1007/978-3-642-10366-7_25)
59. Wikström, D.: Special soundness in the random oracle model. Cryptology ePrint Archive, Report 2021/1265 (2021). <https://eprint.iacr.org/2021/1265>
60. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. Cryptology ePrint Archive, Report 2022/1010 (2022). <https://eprint.iacr.org/2022/1010>
61. Zhang, F., Maram, D., Malvai, H., Goldfeder, S., Juels, A.: DECO: liberating web data using decentralized oracles for TLS. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020, pp. 1919–1938. ACM Press (Nov 2020)
62. ZKProofs Standards. <https://zkproof.org/> (2022)





# Complete Characterization of Broadcast and Pseudo-signatures from Correlations

Varun Narayanan<sup>1(✉)</sup>, Vinod M. Prabhakaran<sup>2(✉)</sup>, Neha Sangwan<sup>2(✉)</sup>,  
and Shun Watanabe<sup>3(✉)</sup>

<sup>1</sup> Technion, Haifa, Israel

<sup>2</sup> Tata Institute of Fundamental Research, Mumbai, India  
{vinodmp,neha\_010}@tifr.res.in

<sup>3</sup> Tokyo University of Agriculture and Technology, Tokyo, Japan  
shunwata@cc.tuat.ac.jp

**Abstract.** Unconditionally secure broadcast is feasible among parties connected by pairwise secure links only if there is a strict two-thirds majority of honest parties when no additional resources are available. This limitation may be circumvented when the parties have recourse to additional resources such as correlated randomness. Fitzi, Wolf, and Wullschleger (CRYPTO 2004) attempted to characterize the conditions on correlated randomness shared among three parties which would enable them to realize broadcast. Due to a gap in their impossibility argument, it turns out that their characterization is incorrect. Using a novel construction we show that broadcast is feasible under a considerably larger class of correlations. In fact, we realize pseudo-signatures, which are information theoretic counterparts of digital signatures using which unconditionally secure broadcast may be obtained. We also obtain a matching impossibility result thereby characterizing the class of correlations on which three-party broadcast (and pseudo-signatures) can be based. Our impossibility proof, which extends the well-know argument of Fischer, Lynch and Merritt (Distr. Comp., 1986) to the case where parties observe correlated randomness, maybe of independent interest.

**Keywords:** Unconditional security · broadcast · pseudo-signatures · information theory

## 1 Introduction

Broadcast is one of the more fundamental primitives in cryptography. For instance, to realize unconditionally secure multiparty secure computation, a strict

---

VN was supported by ISF Grants 1709/14 & 2774/20 and ERC Project NTSC (742754). VP was supported by SERB through project MTR/2020/000308 and DAE under project no. RTI4001. NS was supported by the TCS Foundation through the TCS Research Scholar Program and by DAE under project no. RTI4001. SW is supported in part by the Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant 20H02144.

majority of honest parties suffices if broadcast is available, while, in its absence, more than two-thirds (a supermajority) of the parties must be honest. In fact, in a landmark paper, Lamport, Shostak, and Pease [24] showed that unconditionally secure broadcast can be realized if and only if there is an honest supermajority. Furthermore, they showed that broadcast can be realized with any number of malicious parties if digital signature is available among the parties.

Digital signatures based on public-key cryptography is necessarily only computationally secure. Towards realizing unconditionally secure broadcast without an honest supermajority, Pfitzmann and Waidner [32] introduced the concept of pseudo-signatures. Unlike digital signatures, pseudo-signatures have a fixed *transferability* – the number of transfers that can be made among the parties before they stop being secure. Unconditionally secure broadcast protocols can be realized without an honest supermajority if pseudo-signatures with sufficiently large transferability is available.

In the absence of an honest supermajority, pseudo-signatures (and broadcast) may be realized if the parties have access to correlated random variables. Realizing cryptographic primitives based on such a model has received considerable attention. For instance, secret key generation [1, 20, 25, 35, 37], authentication [26–28, 36], and oblivious transfer [2, 8–10, 12, 23, 31, 33, 39] have all been studied in this model. For broadcast and pseudo-signatures, such a study was initiated by Fitzi, Wolf, and Wullschlegler [19]; see also [14–17].

Pseudo-signature and broadcast protocols using correlated random variables available to the parties were presented in [19]. The necessary and sufficient condition for the correlations for pseudo-signature (and broadcast) to be feasible was also claimed. However, there was a gap in the proof of necessity. In fact, it turns out that the condition itself is not necessary and there are counterexamples. In order to resolve this, novel ideas for both the construction and impossibility are needed which we present here.

There has been a renewed interest in the problem of broadcast (byzantine agreement) because of its connections to blockchains; e.g., see [5] and references therein. In this context, establishing the theoretical foundations of broadcast and pseudo-signatures takes on added interest.

The focus of this paper is on three-party broadcast (and, hence, also on three-party pseudo-signatures). However, there are implications for more than three parties. For instance, it is known that for  $n$  parties, if broadcast among every triple of them is available, then broadcast among all  $n$  parties tolerating  $t < n/2$  corrupted parties is feasible [18] (and hence  $n$ -party MPC with unconditional security with the same threshold for corruption is feasible [3, 34]). We leave the problem of establishing the necessary and sufficient conditions on the correlations to realize broadcast/pseudo-signature for more than three parties as a fascinating open question.

## 1.1 Problem Formulation

We consider a network consisting of three parties,  $P_1$ ,  $P_2$ , and  $P_3$ . The parties are connected by pairwise secure channels; we assume that the network is

synchronous, i.e., each party can recognize who should communicate next based on a common clock. As we mentioned, it is impossible to realize broadcast or pseudo-signatures from scratch if one among the three parties may be malicious. We assume that  $P_1, P_2$ , and  $P_3$  observe random variables  $(X_i)_{i \in [n]}, (Y_i)_{i \in [n]}, (Z_i)_{i \in [n]}$ , respectively, where  $[n] = \{1, 2, \dots, n\}$ , such that the triples  $(X_i, Y_i, Z_i)_{i \in [n]}$ , are independent and identically distributed (i.i.d.) according to a known distribution  $P_{XYZ}$ .

**Broadcast.** In a broadcast protocol, the sender  $P_1$  has a message  $b \in \{0, 1\}$  it wants to convey to the other parties. The parties communicate interactively over the pairwise secure channels. In each round of communication, the communicating party computes the message it sends based on its observations (i.e., parts of the correlation it observes), its transcript so far, and its private randomness. At the end of the protocol, the receivers  $P_2$  and  $P_3$  output  $b_2 \in \{0, 1\}$  and  $b_3 \in \{0, 1\}$ , respectively. We say that a protocol is an  $\varepsilon$ -secure implementation of broadcast if the following two conditions are satisfied:

- *Correctness:* When the sender  $P_1$  is honest with input  $b \in \{0, 1\}$ , then all the honest receivers output  $b$  with probability at least  $1 - \varepsilon$ ;
- *Agreement:* When both the receivers  $P_2$  and  $P_3$  are honest, they output the same value  $b_2 = b_3$  with probability at least  $1 - \varepsilon$ .

We are interested in the necessary and sufficient condition on the distribution  $P_{XYZ}$  of the correlation such that an  $\varepsilon$ -secure implementation of broadcast exists for an arbitrary  $\varepsilon > 0$  and sufficiently many copies  $n$  of the correlation.

**Pseudo-signature.** In a pseudo-signature (PS) protocol with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$ , the sender  $P_1$  has an input message  $b \in \{0, 1\}$ , and the protocol consists of two phases, the signing phase and the transfer phase. In the signing phase, after some prescribed rounds of communication over the pairwise secure channels, the intermediate party  $P_2$  outputs  $b_2 \in \{0, 1, \perp\}$ , where the symbol  $\perp$  indicates that  $P_2$  rejects the message sent by  $P_1$ ; the protocol proceeds to the transfer phase only when  $P_2$  does not reject in the signing phase. In the transfer phase, after some prescribed rounds of communication over the pairwise secure channels, the receiver  $P_3$  outputs  $b_3 \in \{0, 1, \perp\}$ , where the symbol  $\perp$  indicates that  $P_3$  rejects the message transferred by  $P_2$ . We say that a protocol is an  $\varepsilon$ -secure implementation of pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  if the following four conditions are satisfied:

- *Correctness:* If  $P_1$  and  $P_2$  are honest, then  $b_2 = b$  with probability at least  $1 - \varepsilon$ ;
- *Unforgeability:* If  $P_1$  and  $P_3$  are honest, then the probability of the event  $b_3 \notin \{b, \perp\}$  is less than  $\varepsilon$ ;
- *Transferability:* If  $P_2$  and  $P_3$  are honest, then the probability of the event  $(b_2 \neq \perp) \wedge (b_3 \neq b_2)$  is less than  $\varepsilon$ ;

- *Secrecy*: If  $P_1$  and  $P_2$  are honest, then the view of  $P_3$  in the signing phase is almost independent of the message  $b$ , i.e., the conditional distribution of  $P_3$ 's view in the signing phase given  $b = 0$  and  $b = 1$  are close to each other in total variation distance<sup>1</sup>.

We are interested in the necessary and sufficient condition on  $P_{XYZ}$  such that an  $\varepsilon$ -secure pseudo-signature protocol with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  exists for an arbitrary  $\varepsilon > 0$  and sufficiently large  $n$ .

**Relation Between Broadcast and Pseudo-signature.** Broadcast and pseudo-signature are closely related. If a pseudo-signature protocol with either transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  or  $P_1 \rightarrow P_3 \rightarrow P_2$  is available, we can construct a secure implementation of broadcast with sender  $P_1$ ; e.g., see [16, 32]<sup>2</sup>. Thanks to this result, a construction for pseudo-signature implies a construction for broadcast, and an impossibility result for broadcast implies an impossibility result for pseudo-signature. Thus, we focus on constructions of pseudo-signatures and impossibility results for broadcast in this paper. The conditions on correlations under which pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  is feasible and broadcast with sender  $P_1$  is feasible turn out to be same giving us a complete characterization.

## 1.2 Related Work

The problem of the broadcast was introduced in [24], where the necessary and sufficient condition for feasibility of broadcast was established. An alternative proof for the impossibility part was proposed in [13]. This argument has been widely used for proving impossibility results in distributed computing; e.g., see [6, 15]. Many impossibility results on distributed computing have been derived using variants of this argument.

The concept of the pseudo-signature was introduced in [32]; see also [4] for an earlier attempt at introducing an information theoretic version of digital signature. They proposed a pseudo-signature protocol for a one bit message. More efficient constructions based on bivariate polynomials were introduced in [21] (see also [22]).

The problem of implementing broadcast and pseudo-signature from correlated random variables was studied in [19]; this problem was motivated by an implementation of broadcast from measurement outcomes of quantum entanglement [14]. The minimum requirement for realizing broadcast was studied in [15]; for instance, the global broadcast among all the parties can be constructed from the three party broadcast as long as honest majority is satisfied [18].

<sup>1</sup> Our construction provides a protocol with perfect secrecy, while we prove our impossibility results without making use of the secrecy condition. Thus, the secrecy condition does not affect the characterization.

<sup>2</sup> On the other hand, if broadcast protocols with *each* party as sender is available, a pseudo-signature protocol, also known as an information checking protocol, can be constructed [7].

### 1.3 Main Contributions and Results

The main technical contributions of this paper are three-fold:

- (i) We give a construction for pseudo-signature from correlations (Theorem 5) which considerably expands the class of correlations for which it was known to be feasible. As we show, this construction, when combined with (ii) below, fully characterizes the class of feasible correlations which was an open problem [38, Section 18.7].
- (ii) Given a pseudo-signature protocol with transfer path  $P_1 \rightarrow P_3 \rightarrow P_2$ , we construct a pseudo-signature protocol with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  (Theorem 6); an observation which is new to the best of our knowledge.
- (iii) We prove an impossibility for broadcast (which implies tight results for pseudo-signatures and broadcast) by generalizing the well-known technique of Fischer, Lynch, and Merritt [13] to our case which has a setup in the form of correlations at the parties (Theorem 7). We believe this may be of more general interest.

Using these we precisely characterize the class of correlations on which pseudo-signatures (Theorem 8) and broadcast (Theorem 9) can be based. As mentioned earlier, this class is the same for broadcast from sender  $P_1$  and for pseudo-signatures with either of the transfer paths which start with  $P_1$ . We also provide characterizations of the correlations under which pseudo-signature can be obtained when there is limited connectivity/directionality of links between the parties (Theorems 10–12)<sup>3</sup>.

### 1.4 Technical Overview

**Pseudo-signature from Correlations.** Attempts at altering a random variable  $X$  may leave a statistical trace that a party who holds a correlated random variable  $Y$  may be able to detect – this observation forms the basis of building a pseudo-signature protocol from correlation. The party with  $Y$  can only hope to detect anomalies in those parts of  $X$  which non-trivially depend on  $Y$ . For instance, if two symbols  $x$  and  $x'$  satisfy  $P_{Y|X}(\cdot|x) = P_{Y|X}(\cdot|x')$ , then a party observing  $Y$  cannot detect the swapping of  $x$  and  $x'$ . To account for this, following [19], let us define  $X \searrow Y$  as the maximal part of  $X$  which non-trivially depends on  $Y$  (in statistics, this is known as the minimal sufficient statistic for  $Y$  given  $X$ ).

**Definition 1 (Minimal sufficient statistics).** *For a pair of random variables  $X, Y$  with joint distribution  $P_{XY}$ , consider the partition of the alphabet  $\mathcal{X}$  of  $X$  induced by the following equivalence relation:*

$$x \sim x' \text{ if } P_{Y|X}(y|x) = P_{Y|X}(y|x'), \forall y.$$

*We define  $\psi_{X \searrow Y}$  to be the function which maps the elements in  $\mathcal{X}$  to their cell (i.e., part) in the above partition, and we define  $X \searrow Y \stackrel{\text{def}}{=} \psi_{X \searrow Y}(X)$ .*

---

<sup>3</sup> A similar question for broadcast turns out to be trivial.

Notice that  $X \setminus Y$  is a random variable which is a function of  $X$  alone, where the function  $(\psi_{X \setminus Y})$  is defined in terms of the distribution  $P_{XY}$ .

Fitzi, Wolf, and Wullschleger [19] constructed a pseudo-signature protocol for transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  from  $n$  i.i.d. copies of a correlation  $X, Y, Z$  observed at parties  $P_1, P_2, P_3$ , respectively. The sender  $P_1$  attaches copies of  $X \setminus Y$  as the signature (the first  $n/2$  copies for message bit 0 and the second  $n/2$  for message bit 1). As foreshadowed,  $P_2$ , who holds the copies of  $Y$ , may detect any significant anomalies in the signature (with high confidence for sufficiently large  $n$ ; see Lemma 1). They showed that their protocol is secure as long as the simulatability condition<sup>4</sup>

$$X \setminus Y \longleftrightarrow Y \longleftrightarrow Z \tag{1}$$

does *not* hold. Heuristically, (1) says that  $P_2$  can forge  $P_1$ 's signature  $X \setminus Y$  using  $Y$  in a manner undetectable by  $P_3$  relying on its observations  $Z$ .

Furthermore, it was claimed in [19] that when  $X \setminus Y \longleftrightarrow Y \longleftrightarrow Z$  and  $X \setminus Z \longleftrightarrow Z \longleftrightarrow Y$  hold, then pseudo-signature with either of the transfer paths in which  $P_1$  is the sender (i.e.,  $P_1 \rightarrow P_2 \rightarrow P_3$  or  $P_1 \rightarrow P_3 \rightarrow P_2$ ) is impossible to build based on  $(X, Y, Z)$ . However, there is a gap in the proof of [19, Theorem 3]<sup>5</sup>, and a pseudo-signature can be implemented even if both these Markov chains hold. In essence, these simulatability conditions do not account for the fact that  $P_2$  and  $P_3$  may effectively “upgrade” their observations by communicating with the other parties even if they do not trust the other parties. We demonstrate different possible ways in which this can be accomplished using a few examples; these examples also serve as counterexamples to [19, Theorem 3] and build up intuition for our construction.

*Example 1 (Upgrading  $P_3$ 's observation by communication from  $P_1$ ).* Let us consider a correlation induced by Rabin's oblivious transfer. Suppose that  $X$  is uniformly distributed on  $\{0, 1\}$ , and  $Y = Z$  such that it is  $X$  with probability  $\frac{1}{2}$  and the erasure symbol  $\mathbf{e}$  with probability  $\frac{1}{2}$ . Then, since  $Y = Z$ , this correlation satisfies both  $X \setminus Y \longleftrightarrow Y \longleftrightarrow Z$  and  $X \setminus Z \longleftrightarrow Z \longleftrightarrow Y$ . However, we can implement a pseudo-signature protocol from this correlation as follows. Let  $(X_i, Y_i, Z_i)_{i \in [n]}$  be i.i.d. observations distributed according to  $P_{XYZ}$ .

1. To send message  $b \in \{0, 1\}$ ,  $P_1$  sends  $b$  and  $(\tilde{X}_i)_{i \in T_b} = (X_i)_{i \in T_b}$  to  $P_2$ , where  $T_b = \{i : \frac{bn}{2} + 1 \leq i \leq \frac{(b+1)n}{2}\}$ .
2.  $P_2$  rejects if  $Y_i \notin \{X_i, \mathbf{e}\}$  for some  $i \in T_b$ . Otherwise,  $P_2$  accepts  $b$ .
3.  $P_1$  sends  $(\hat{X}_i)_{i \in [n]} = (X_i)_{i \in [n]}$  to  $P_3$ .
4. To transfer a message  $b$  it accepted,  $P_2$  sends  $\hat{b} = b$  and  $(\check{X}_i)_{i \in T_b} = (\tilde{X}_i)_{i \in T_b}$  to  $P_3$ .

<sup>4</sup> We write “the Markov chain  $U \longleftrightarrow V \longleftrightarrow W$  holds,” or simply “ $U \longleftrightarrow V \longleftrightarrow W$ ” to mean  $U$  and  $W$  are conditionally independent conditioned on  $V$ .

<sup>5</sup> It turns out that by considering restricted connectivity, specifically, when there is no link between  $P_1$  and  $P_3$  and the link from  $P_2$  to  $P_3$  is unidirectional, it can be shown that pseudo-signature from  $(X, Y, Z)$  is possible (if and) only if  $X \setminus Y \longleftrightarrow Y \longleftrightarrow Z$  is not a Markov chain (see Theorem 12).

5.  $P_3$  accepts  $\hat{b}$  if any one of the following holds: (i)  $Z_i \notin \{\hat{X}_i, \mathbf{e}\}$  for some  $i \in [n]$ ; and (ii)  $|\{i \in T_{\hat{b}} : \check{X}_i \neq \hat{X}_i\}| \leq n\delta$  for a parameter  $\delta > 0$ . Otherwise,  $P_3$  rejects.

Clearly, the above protocol is perfectly correct. To verify transferability (security against  $P_1$ ), notice that, to be successful, a corrupt  $P_1$  must convince  $P_3$  to reject in step 5. while ensuring that  $P_2$  accepts in step 2.. In order for this,  $P_1$ 's transmissions  $(\check{X}_i)_{i \in T_b}$  and  $(\hat{X}_i)_{i \in T_b}$  to  $P_2$  and  $P_3$ , respectively, must disagree in more than  $n\delta$  locations. However, unless the observations of  $P_2$  and  $P_3$  (i.e.,  $Y_i = Z_i$ ) are not  $\mathbf{e}$  in *all* such locations, the attack will not succeed (since either  $P_2$  will reject in step 2. or  $P_3$  will accept the bit  $P_2$  transfers in step 5.). Hence, the chance of success for  $P_1$  is at most  $2^{-\delta n}$ . To see unforgeability (security against  $P_2$ ), notice that a successful attack by  $P_2$  requires it to guess  $X_i$  for all  $i \in T_{1-b}$  where  $Y_i = \mathbf{e}$  such that the number of incorrect guesses is no more than  $n\delta$ . Since for each  $i \in T_{1-b}$ , the probability of  $Y_i = \mathbf{e}$  is  $1/2$  and  $P_2$  has even odds of guessing correctly, the probability of a successful attack is  $2^{-\Omega(n)}$  for  $\delta < 1/8$  (see, e.g., [29, Theorem 4.5]). ▷

In the above example,  $P_3$ 's observation is effectively upgraded from  $Z$  to  $(X, Z)$  using communication from  $P_1$  in step 3.. Note that (in step 5.)  $P_3$  verifies this communication from  $P_1$ ; if the verification fails,  $P_3$  accepts any message  $P_2$  transfers, and if the verification succeeds, with overwhelming probability  $P_1$  has not lied on more than a small fraction of locations. In general,  $P_3$  can only (statistically) verify the parts of  $X$  which non-trivially depend on  $Z$ , namely  $X \searrow Z$  (which happens to be  $X$  in this example). Thus, following the intuition in the example, we may build<sup>6</sup> a pseudo-signature protocol from correlations  $(X, Y, Z)$  whenever the following condition (which is stronger compared to (1)) does not hold:

$$X \searrow Y \iff Y \iff (Z, X \searrow Z). \tag{2}$$

It turns out that we may further expand the class of feasible correlations by upgrading  $P_3$ 's observation via communication from both  $P_1$  and  $P_2$  (also see [30, Example 3 in Appendix A]). Note that the communication from  $P_2$  could be part of the transfer step (i.e., step 4. in the example), where now, in addition to what was received from  $P_1$ , party  $P_2$  may also send its observation  $Y$  to  $P_3$ . Assume that, as in the example,  $P_1$  sends its  $X$  observations and  $P_3$  has verified the  $X \searrow Z$  part of this (if the verification fails,  $P_3$  accepts the message transferred by  $P_2$ ). Now,  $P_3$  may also verify the part  $Y \searrow Z$  of the  $Y$  observations sent by  $P_2$ . If the verification fails,  $P_3$  rejects the message  $P_2$  transfers. Otherwise, it may now upgrade its observation to  $Z_1 = (Z, (X \searrow Z), (Y \searrow Z))$ . Using this  $P_3$  is now able to verify more parts of the  $X$  observations received from  $P_1$ , specifically,  $X \searrow Z_1$ , and similarly  $Y \searrow Z_1$  of the  $Y$  received from  $P_2$ . It is clear that this procedure can be repeated. In each step, the upgrade operation involves a verification step where, based on what is currently known, the maximal dependent part of the

---

<sup>6</sup> This needs a slightly more elaborate test from the one in the protocol in the example which exploits the fact that Rabin OT has erasures and no “errors.”

observation being used to perform the upgrade is verified. If the verification fails,  $P_3$  either accepts or rejects the transferred message depending on who provided the observation. It turns out that only a fixed number of such steps suffice to reach the best possible upgraded observation. The number of steps needed depends on the distribution of  $(X, Y, Z)$  and is at most  $|\mathcal{X}||\mathcal{Y}|$ , the product of the alphabet sizes of  $X$  and  $Y$ ; see Definition 3, the discussion after that, and Definition 4. We denote by  $X^\ddagger$  (see Definition 4) the additional information  $P_3$  acquires about  $X$  through this repeated upgradation procedure. The above intuition allows building a pseudo-signature protocol whenever the following condition (which is stronger still compared to (2)) does not hold:

$$X \setminus Y \longleftrightarrow Y \longleftrightarrow (Z, X^\ddagger). \tag{3}$$

In some cases it is also useful to upgrade  $P_2$ 's observation partially so that the party is able to verify  $P_1$ 's signature, but is still unable to forge the signature. The following example illustrates the idea.

*Example 2 (Partially upgrading  $P_2$ 's observation using communication from  $P_3$ ).* Consider a function MAC that takes a message  $M$  and a key  $K$  and computes an information theoretic message authentication code (MAC) as  $\text{MAC}_K(M)$ . MACs guarantee non-forgability, i.e., a forger (who may have access to a (message, MAC) pair) can succeed in producing a valid MAC for a fresh message with negligible probability if the key is unknown. Let MAC and  $\text{MAC}'$  be two information theoretic MACs such that MAC takes a message from  $\mathcal{U}$  and a key uniformly at random from  $\mathcal{V}$ , and  $\text{MAC}'$  takes a message from  $\mathcal{V}$  and a key uniformly at random from  $\mathcal{W}$ . Consider the correlation  $(X, Y, Z)$  generated using the following process:

1. Sample  $U, V, W$  uniformly and independently from  $\mathcal{U}, \mathcal{V}, \mathcal{W}$ , respectively.
2. Define  $X = (U, \text{MAC}_V(U))$ ,  $Y = W$  and  $Z = (V, \text{MAC}'_W(V))$ .

For all  $i \in [m], j \in \{0, 1\}$ , suppose  $(X_{i,j}, Y_{i,j}, Z_{i,j})$  are i.i.d. according to the distribution of  $(X, Y, Z)$  described above. When  $P_1, P_2$  and  $P_3$  have  $(X_{i,j})$ ,  $(Y_{i,j})$  and  $(Z_{i,j})$ , respectively, for  $i \in [m], j \in \{0, 1\}$ , the following protocol implements pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$ .

**Signing Phase**

1.  $P_3$  choose  $S \subset [m]$  uniformly at random conditioned on  $|S| = \frac{m}{2}$  and sends  $S$  and  $(\hat{Z}_{i,j})_{i \in S, j \in \{0,1\}} = (Z_{i,j})_{i \in S, j \in \{0,1\}}$  to  $P_2$ .
2. To send message  $b \in \{0, 1\}$ ,  $P_1$  sends  $b$  and  $(\tilde{X}_{i,b})_{i \in [m]} = (X_{i,b})_{i \in [m]}$  to  $P_2$ .
3.  $P_2$  accepts  $b$  unless for some  $i \in S$ ,  $\tilde{X}_{i,b} = (u, \bar{u}), Y_{i,b} = w, \hat{Z}_{i,b} = (v, \bar{v})$  such that  $\bar{u} \neq \text{MAC}_v(u)$  and  $\bar{v} = \text{MAC}'_w(v)$ .

**Transfer Phase**

4. To transfer a message  $b$  it accepted,  $P_2$  sends  $\hat{b} = b$  and  $(\check{X}_{i,\hat{b}})_{i \in [m]} = (\tilde{X}_{i,b})_{i \in [m]}$  to  $P_3$ .
5.  $P_3$  rejects  $\hat{b}$  if there are more than  $m/4$  distinct  $i \in [m] \setminus S$  such that  $\bar{u}_i \neq \text{MAC}_{v_i}(u_i)$ , where  $\check{X}_{i,\hat{b}} = (u_i, \bar{u}_i)$  and  $Z_{i,\hat{b}} = (v_i, \bar{v}_i)$ . Otherwise,  $P_3$  accepts  $\hat{b}$ .



To see *security against*  $P_1$  (transferability), notice that a corrupt  $P_1$  must ensure that the  $(\tilde{X}_{i,b})_{i \in [m]} = (u_i, \bar{u}_i)_{i \in [m]}$  it sends to  $P_2$  is such that for all  $i \in S$ ,  $\bar{u}_i = \text{MAC}_{V_{i,b}}(u_i)$  so that  $P_2$  accepts and for at least  $m/4$  of the remaining  $i \in [m] \setminus S$ ,  $\bar{u}_i \neq \text{MAC}_{V_{i,b}}(u_i)$  so that  $P_3$  rejects. Since  $S$  is a set unknown to  $P_1$  of size  $m/2$  chosen uniformly at random from  $[m]$ , the probability of success is negligible in  $m$  (i.e.,  $2^{-\Omega(m)}$ ). *Security against*  $P_2$  (unforgeability) follows from the security of MAC. To convince  $P_3$  to accept  $\hat{b} = 1 - b$ , the corrupt  $P_2$  needs to generate  $m/2$  purported (message, MAC) pairs corresponding to keys  $(V_{i,1-b})_{i \in [m] \setminus S}$  (which it does not know) such that at least  $m/4$  of them are valid pairs. Using a MAC with security parameter  $\epsilon/m$ , we may ensure that only with at most  $\epsilon$  probability will even one of the pairs be valid. Unlike the previous examples,  $P_3$  participates in the signing phase. Hence we also need to consider *security against*  $P_3$  (correctness). This will follow from the security of  $\text{MAC}'$ . A corrupt  $P_3$  (suppose it correctly guessed  $P_1$ 's  $b$ ), who observes  $Z_{i,b} = (V_{i,b}, \text{MAC}'_{W_{i,b}}(V_{i,b}))$ ,  $i \in [m]$ , needs to generate a valid (message, MAC) pair  $(v'_{i,b}, \bar{v}'_{i,b})$  corresponding to the (unknown) key  $W_{i,b}$  such that  $v'_{i,b} \neq V_{i,b}$  for at least one  $i \in [m]$  so that (with  $S \ni i$ ),  $P_2$  may reject  $b$  if  $\text{MAC}_{V_{i,b}}(U_{i,b}) \neq \text{MAC}_{v'_{i,b}}(U_{i,b})$ . If  $\text{MAC}'$  has security parameter  $\epsilon/m$ , the probability that  $P_3$  succeeds is at most  $\epsilon$ . Hence, the protocol is  $\epsilon$ -secure if  $m = O(\log(1/\epsilon))$  and the MACs are  $(\epsilon/m)$ -secure.  $\triangleright$

In the above example, by providing  $P_3$ 's observation (i.e.,  $Z$ ) to  $P_2$  on a random subset  $S$  of the instances, we achieved two things. Firstly, this enabled  $P_2$  to verify the signature sent by  $P_1$  with the same confidence as  $P_3$  would on these instances. Since  $S$  is unknown to  $P_1$ , if  $P_2$  does not detect an anomaly among these, with overwhelming probability,  $P_1$  has not lied on more than a constant fraction of all the instances. Secondly, by the independence of these instances,  $P_2$  is still as oblivious about  $P_3$ 's observation outside of  $S$  as before the upgrade.  $P_3$  checking the signature only on  $[m] \setminus S$  denies  $P_2$  the possibility of a forging attack.

Note that a corrupt  $P_3$  could try to make  $P_2$  distrust an honest  $P_1$  by giving out incorrect values of its observation (on a potentially cherry-picked set  $S$ ; in the above example there was no advantage in cherry picking  $S$ ). Hence, it is important that  $P_2$  uses only those parts of  $P_3$ 's observation it can verify to be correct, i.e.,  $Z \setminus Y$  (in the example, this component turns out to be all of  $P_3$ 's observation). If the verification fails,  $P_2$  accepts  $P_1$ 's message. In general, these verifications may require a statistical test which may be reliable only when run over a long vector of observations (in the example, MAC allowed this verification to be done element-wise); the same is true for (the upgraded)  $P_2$  verifying  $P_1$ 's signature (which was again possible element-wise in the example). Thus, in our construction, we consider two indices:  $i \in [m]$  which serves the same purpose as in the above example and  $j \in [n]$  such that in step 2, for each  $i \in S$ , statistical tests can be carried out by  $P_2$  over a vector indexed by  $j$ ; such tests also takes away any advantage  $P_3$  can hope to gain by picking  $S$  carefully. Similarly, in step 5, for each  $i \in [m] \setminus S$ ,  $P_3$  conducts statistical tests over vectors indexed by  $j$ . With these we may obtain a pseudo-signature protocol construction in which (the upgraded)  $P_2$  can verify signatures  $X \setminus Y'$ , where  $Y' := (Y, Z \setminus Y)$ .

This pseudo-signature protocol is secure as long as the following is not a Markov chain (a stronger condition than (1)):

$$X \setminus Y' \longleftrightarrow Y \longleftrightarrow Z.$$

Notice that only the first random variable has changed in the condition. The middle random variable continues to be  $Y$  (and not  $Y'$ ). This is because the upgrade of  $P_2$ 's observation to  $Y'$  is limited to a random subset  $S$ , and, as we saw in the example,  $P_3$  verifies the signature transferred by  $P_2$  in  $[m] \setminus S$  where  $P_2$  only holds  $Y$ .

Now suppose  $P_2$  has verified  $X \setminus Y'$  as above and hence holds  $Y'' = (Y', X \setminus Y')$ . Then,  $P_2$  may further upgrade its observation by verifying more of the  $Z$  it received from  $P_3$ . In particular,  $P_2$  may verify  $Z \setminus Y''$  and, if the verification passes (in case of failure  $P_2$  accepts  $P_1$ 's message),  $P_2$  may upgrade itself to  $Y''' = (Y'', Z \setminus Y'')$ . With this additional upgrade,  $P_2$  is equipped to verify a heftier signature (specifically  $X \setminus Y'''$ ). It is clear that this procedure may be repeated, similar to the repeated upgradation procedure we saw for  $P_3$ . In each alternate step,  $P_2$  verifies  $Z$  and  $X$  until no further upgradation is possible (this is again attained in only a finite number of steps). We denote by  $X^\dagger$  (see Definition 5) the part of  $X$  that the repeated upgradation procedure allows  $P_2$  to verify and learn. We emphasize that  $P_2$  learns  $X^\dagger$  only on a subset of instances which prevents it from using this information to mount a successful forging attack. Thus, pseudo-signature is feasible if the following Markov chain does not hold:

$$X^\dagger \longleftrightarrow Y \longleftrightarrow Z. \tag{4}$$

Our construction for pseudo-signature in Sect. 3.1, which combines all the ideas above and upgrades the observations of  $P_3$  and  $P_2$  (in a subset), gives the following result (cf. (3)–(4)):

**Theorem 1 (informal).** *Pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  from correlation  $(X, Y, Z)$  is feasible if the following is not a Markov chain*

$$X^\dagger \longleftrightarrow Y \longleftrightarrow (Z, X^\ddagger). \tag{5}$$

**Characterization of Correlations.** We also observe that given a pseudo-signature protocol with transfer path  $P_1 \rightarrow P_3 \rightarrow P_2$ , we may construct a pseudo-signature protocol with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$ . Our construction in Sect. 3.2 effectively samples a correlation using the given pseudo-signature protocol and uses this correlation to implement the protocol with the requisite altered transfer path. This observation, when combined with Theorem 1, implies that pseudo-signature from a correlation  $(X, Y, Z)$  is feasible if either (5) or its analog when the roles of  $Y$  and  $Z$  are swapped does not hold. The latter Markov chain is in fact  $X^\ddagger \longleftrightarrow Z \longleftrightarrow (Y, X^\dagger)$ , i.e., exchanging the roles of  $Y$  and  $Z$  also exchanges the corresponding upgrades  $X^\dagger$  and  $X^\ddagger$  as is evident from the similarity of the repeated upgradation procedures at  $P_2$  and  $P_3$  (also see Definitions 3–5). This gives the construction for our characterization theorem for pseudo-signatures.

**Theorem 2 (informal).** *If parties  $P_1, P_2, P_3$  observe independent copies of correlation  $(X, Y, Z)$ , a pseudo-signature protocol with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  exists if and only if at least one of the following Markov chains does not hold:*

$$X^\dagger \longleftrightarrow Y \longleftrightarrow (Z, X^\ddagger) \quad (6)$$

$$X^\ddagger \longleftrightarrow Z \longleftrightarrow (Y, X^\dagger) \quad (7)$$

**Impossibility.** To show the impossibility part of the above theorem, we leverage the connection between pseudo-signatures and broadcast. As mentioned earlier, broadcast with sender  $P_1$  may be realized using a pseudo-signature protocol with  $P_1$  as the signing party. We show that broadcast from correlation  $(X, Y, Z)$  with sender  $P_1$  is impossible if both Markov chains (6) and (7) hold. Our impossibility proof in Sect. 4 is along the lines of the proof of impossibility of three-party broadcast from scratch due to Fischer, Lynch, and Merritt (FLM) [13]. It may be thought of as an extension of their argument to the case when correlations are available to the parties. Similar to [13], we make two copies of the parties (in fact, only copying party  $P_1$  suffices) and rewire the parties to create a fictitious network. The parties in this network are fed observations from a *carefully chosen correlation* so that we may give three different interpretations which lead to a contradiction. Under each interpretation, two of the parties are honest and the third dishonest party simulates the remaining parties in the rewired network. Moreover, the choice of correlation fed to the parties in the rewired network is such that in each interpretation the correlations of the two honest parties and the one dishonest party are  $(X, Y, Z)$ , and the dishonest party is able to sample the correlations needed to perform the simulation. Like in [13], the interpretations lead to a contradiction proving the impossibility. We note here that the rewired network we use is identical to the one in the (flawed) proof of [19, Theorem 7], however the rest of the proof including our use of a carefully chosen distribution is different. To the best of our knowledge this is the first instance where FLM’s argument has been extended to a problem with setup where a carefully chosen setup is provided to the parties in the fictitious network. In [6], the FLM argument was applied to a case where parties may invoke partial broadcast (e.g., three party broadcast); there, providing the parties in the fictitious network with (the more obvious choice of) partial broadcast sufficed. Given the extensive use of FLM argument in proving impossibility results in distributed computing, our extension might be of independent interest. From the above discussion it is clear that we also have a characterization theorem for broadcast from correlations:

**Theorem 3 (informal).** *If parties  $P_1, P_2, P_3$  observe independent copies of correlation  $(X, Y, Z)$ , broadcast with sender  $P_1$  is feasible if and only if at least one of the following Markov chains does not hold:*

$$X^\dagger \longleftrightarrow Y \longleftrightarrow (Z, X^\ddagger)$$

$$X^\ddagger \longleftrightarrow Z \longleftrightarrow (Y, X^\dagger)$$

**Pseudo-signature Under Limited Connectivity.** Our construction only used the unidirectional links  $P_1 \Rightarrow P_2$ ,  $P_1 \Rightarrow P_3$ , and the bidirectional link  $P_2 \Leftrightarrow P_3$ , while the impossibility applies for protocols which may use all pairwise links in either direction. We also study pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$ , which necessarily requires the  $P_1 \Rightarrow P_2$  and  $P_2 \Rightarrow P_3$  links, when one or both of  $P_1 \Rightarrow P_3$  and  $P_3 \Rightarrow P_2$  links that our construction additionally used are absent and provide the characterizations.

## 2 Preliminaries

We use the method of types from information theory [11] to prove some of the technical lemmas; see [30, Appendix B.1] for a review. A sequence  $(x_i)_{i \in [n]}$  is said to be *typical* with respect to a distribution  $P_X$  when its empirical probability is close to  $P_X$  as below:

**Definition 2 ( $\gamma$ -typical sequences).** Let  $X$  be a random variable defined on alphabet  $\mathcal{X}$  with distribution  $P_X$ . For a parameter  $\gamma > 0$ , a sequence  $(x_i)_{i \in [n]}$ , where  $x_i \in \mathcal{X}, i \in [n]$ , is said to be (strongly)  $\gamma$ -typical if for all  $a \in \mathcal{X}$  and for  $N(a, (x_i)_{i \in [n]}) \stackrel{\text{def}}{=} |\{i : x_i = a, i \in [n]\}|$ ,

$$\left| \frac{N(a, (x_i)_{i \in [n]})}{n} - P_X(a) \right| \leq \frac{\gamma}{|\mathcal{X}|}.$$

The set of all  $\gamma$ -typical sequences is denoted by  $\mathcal{T}_\gamma^n(P_X)$ .

The nomenclature is justified by the following theorem [11, Lemma 2.12] which states that when  $(X_i)_{i \in [n]}$  is drawn i.i.d. according to  $P_X$ , with overwhelming probability it will fall in  $\mathcal{T}_\gamma^n(P_X)$ .

**Theorem 4.** Let  $(X_i)_{i \in [n]}$  be a sequence of i.i.d.  $P_X$  random variables, then for any  $0 < \gamma \leq 1/2$ ,

$$\Pr [(X_i)_{i \in [n]} \notin \mathcal{T}_\gamma^n(P_X)] = 2^{-\Omega(n)}. \tag{8}$$

We now formalize the intuition we gave for minimal sufficient statistics  $X \setminus Y$  (see the discussion around Definition 1). The following lemma states that if  $(X_i, Y_i)_{i \in [n]}$  is i.i.d. according to  $P_{XY}$  and a party who possesses  $(X_i)_{i \in [n]}$ , but crucially no additional side-information about  $(Y_i)_{i \in [n]}$ , attempts to tamper with  $(X_i)_{i \in [n]}$  such that the  $\psi_{X \setminus Y}$  parts are significantly altered, with overwhelming probability this attempt can be detected by a second party who possesses the correlated observations  $(Y_i)_{i \in [n]}$ . Proofs of the lemmas in this section are available in [30, Appendix B].

**Lemma 1.** For any joint distribution  $P_{XY}$  and  $\gamma > 0$ , there exists  $\delta > 0$  that approaches 0 as  $\gamma$  approaches 0 such that, when  $(X_i, Y_i)_{j \in [n]}$  are i.i.d. according to  $P_{XY}$ , and  $(\hat{X}_j)_{j \in [n]} \longleftrightarrow (X_j)_{j \in [n]} \longleftrightarrow (Y_j)_{j \in [n]}$  is a Markov chain, then

$$\Pr \left[ (|\{j : \psi_{X \setminus Y}(\hat{X}_j) \neq \psi_{X \setminus Y}(X_j)\}| > n\delta) \wedge ((\hat{X}^n, Y^n) \in \mathcal{T}_\gamma^n(P_{XY})) \right] \leq 2^{-\Omega(n)}. \tag{9}$$

The following lemma states two basic properties of minimal sufficient statistics (Definition 1).

**Lemma 2.** (i)  $X \longleftrightarrow (X \setminus Y) \longleftrightarrow Y$   
 (ii) Suppose  $Y_1$  is a function of  $Y_2$ , then  $X \setminus Y_1 = (X \setminus Y_2) \setminus Y_1$  and hence  $X \setminus Y_1$  is a function of  $X \setminus Y_2$ . i.e., the partition of  $\mathcal{X}$  corresponding to  $X \setminus Y_2$  is a refinement of the one corresponding to  $X \setminus Y_1$ .

The following random variables play a role in the upgrade of  $P_3$ 's observation.

**Definition 3 (Upgraded random variables).** For a triple of random variables  $(X, Y, Z)$  with joint distribution  $P_{XYZ}$ , we define:

$$\begin{aligned} Z^{(1)} &= (Z, (X \setminus Z), (Y \setminus Z)) \\ Z^{(2)} &= (Z^{(1)}, (X \setminus Z^{(1)}), (Y \setminus Z^{(1)})) \\ &\vdots \\ Z^{(i+1)} &= (Z^{(i)}, (X \setminus Z^{(i)}), (Y \setminus Z^{(i)})) \\ &\vdots \end{aligned}$$

For  $j > i$ , note that  $Z^{(i)}$  is a function of  $Z^{(j)}$ . Hence, by Lemma 2(ii),  $(X \setminus Z^{(i)})$  (resp.,  $(Y \setminus Z^{(i)})$ ) is a function of  $(X \setminus Z^{(j)})$  (resp.,  $(Y \setminus Z^{(j)})$ ). In other words, as  $i$  increases,  $X \setminus Z^{(i)}$  and  $Y \setminus Z^{(i)}$  correspond to finer and finer partitions of  $\mathcal{X}$  and  $\mathcal{Y}$ , respectively. Here  $\mathcal{X}$  and  $\mathcal{Y}$  denote the alphabets of  $X$  and  $Y$  respectively. Clearly, if for some  $i$ ,  $((X \setminus Z^{(i+1)}), (Y \setminus Z^{(i+1)})) = ((X \setminus Z^{(i)}), (Y \setminus Z^{(i)}))$  a.s. (i.e., with probability 1), then, for all  $j \geq i$ ,  $Z^{(j)} = Z^{(i)}$  a.s. and  $((X \setminus Z^{(j)}), (Y \setminus Z^{(j)})) = ((X \setminus Z^{(i)}), (Y \setminus Z^{(i)}))$  a.s. Hence, the finest partitions are attained (at least) by index  $i = |\mathcal{X}||\mathcal{Y}|$  and we denote these using the following notation.

**Definition 4 (Upgraded random variables cont.).**

$$X^\dagger = X \setminus Z^{(|\mathcal{X}||\mathcal{Y}|)}, \quad Y^* = Y \setminus Z^{(|\mathcal{X}||\mathcal{Y}|)}.$$

Analogously, for the upgrade of  $P_2$ 's observations, we define:

**Definition 5 (Upgraded random variables cont.).** For a triple of random variables  $(X, Y, Z)$  with joint distribution  $P_{XYZ}$ , we recursively define the following random variables:

$$\begin{aligned} Y^{(1)} &= (Y, (X \setminus Y), (Z \setminus Y)) \\ Y^{(2)} &= (Y^{(1)}, (X \setminus Y^{(1)}), (Z \setminus Y^{(1)})) \\ &\vdots \\ Y^{(i+1)} &= (Y^{(i)}, (X \setminus Y^{(i)}), (Z \setminus Y^{(i)})) \\ &\vdots \\ X^\dagger &= X \setminus Y^{(|\mathcal{X}||\mathcal{Z}|)}, \quad Z^* = Z \setminus Y^{(|\mathcal{X}||\mathcal{Z}|)}. \end{aligned}$$

The following lemma follows from the definitions and Lemma 2(i):

- Lemma 3.** (i)  $X \longleftrightarrow X^\dagger \longleftrightarrow (Y, Z^*)$   
(ii)  $Z \longleftrightarrow Z^* \longleftrightarrow (Y, X^\dagger)$   
(iii)  $X \longleftrightarrow X^\ddagger \longleftrightarrow (Z, Y^*)$   
(iv)  $Y \longleftrightarrow Y^* \longleftrightarrow (Z, X^\ddagger)$

### 3 Constructions

In this section we present our main constructions. In Sect. 3.1 we show that pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  is feasible if the parties  $P_1, P_2, P_3$  observe correlations  $X, Y, Z$ , respectively, such that  $X^\dagger \longleftrightarrow Y \longleftrightarrow (Z, X^\ddagger)$  is not a Markov chain. In Sect. 3.2 we argue that, given a pseudo-signature protocol with transfer path  $P_1 \rightarrow P_3 \rightarrow P_2$ , we can obtain a pseudo-signature protocol with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  (albeit with a weaker security parameter). These two constructions together give us the feasibility direction of the characterizations of correlations which allow pseudo-signatures (and broadcast); see Sect. 5.

#### 3.1 A Pseudo-signature Protocol from Correlations

**Theorem 5.** *Suppose  $P_{XYZ}$  is a joint distribution such that the Markov chain  $X^\dagger \longleftrightarrow Y \longleftrightarrow (Z, X^\ddagger)$  does not hold. Then, for any  $\epsilon > 0$ , there is an  $\epsilon$ -secure pseudo-signature scheme with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  which uses  $N = O(\log^2(\frac{1}{\epsilon}))$  independent copies of the correlation  $(X, Y, Z)$ .*

The construction we use to prove this theorem relies on a statistical test which we describe first: Consider a joint distribution  $P_{UVW}$ . Let  $U^{(0)} = U$ , and  $U^{(r)}$  be recursively defined as (cf. Definitions 3 and 4)

$$U^{(r)} = (U^{(r-1)}, V \setminus U^{(r-1)}, W \setminus U^{(r-1)}), \quad 1 \leq r \leq |\mathcal{V}||\mathcal{W}|.$$

For parameters  $\gamma_1, \dots, \gamma_{|\mathcal{V}||\mathcal{W}|} > 0$  (to be decided; see Lemma 4), the statistical test  $\Sigma_{UVW}$  takes as input  $(u_j, v_j, w_j)_{j \in [n]}$  and proceeds as follows:

1. Set  $u_j^{(0)} = u_j$  for each  $j \in [n]$  and set  $r = 1$ .
2. If  $(u_j^{(r-1)}, v_j)_{j \in [n]} \notin \mathcal{T}_{\gamma_r}^n(P_{U^{(r-1)}, V})$ , report failure w.r.t.  $V$  and terminate.
3. If  $(u_j^{(r-1)}, w_j)_{j \in [n]} \notin \mathcal{T}_{\gamma_r}^n(P_{U^{(r-1)}, W})$ , report failure w.r.t.  $W$  and terminate.
4. Set  $u_j^{(r)} = (u_j^{(r-1)}, \psi_{V \setminus U^{(r-1)}}(v_j), \psi_{W \setminus U^{(r-1)}}(w_j))$  for each  $j \in [n]$ , and set  $r = r + 1$ . If  $r \leq |\mathcal{V}||\mathcal{W}|$ , go to step 2; else report success and terminate.

For  $r = 1$  to  $|\mathcal{V}||\mathcal{W}|$ , the test attempts to recursively “upgrade”  $(u_j^{(r-1)})_{j \in [n]}$  by attaching to it  $(f^{(r)}(v_j), g^{(r)}(w_j))_{j \in [n]}$ . Before doing so, the test must verify if these attachments are valid. The intuition here is that the function  $\psi_{V \setminus U^{(r-1)}}$  of  $(v_j)_{j \in [n]}$  (resp.  $\psi_{W \setminus U^{(r-1)}}$  of  $(w_j)_{j \in [n]}$ ) is indeed something a statistical test

can be used to test the validity of based on  $(u_j^{(r-1)})_{j \in [n]}$  (see Definition 1). Step 2 (resp., 3) performs this validity check by testing whether  $(v_j)_{j \in [n]}$  (resp.,  $(w_j)_{j \in [n]}$ ) is “typical” with the current  $u_j^{(r-1)}$  according to the joint distribution  $P_{U^{(r-1)}, V}$  (resp.,  $P_{U^{(r-1)}, W}$ ). If not, the test terminates at this step with failure w.r.t.  $V$  (resp.  $W$ ). The test terminates with success if none of the validity tests fail. When this happens, the test has verified the validity of  $(\psi_{V \setminus U^{(r-1)|W}}(v_j), \psi_{W \setminus U^{(r-1)|W}}(w_j))_{j \in [n]}$ .

The following lemma formalizes the intuition. It states that while the test will report failure with negligible property when run with inputs  $(U_j, V_j, W_j)_{j \in [n]}$  generated  $P_{UVW}$  i.i.d., it is also robust to maliciously generated inputs. Specifically, if  $(W_j)_{j \in [n]}$  is replaced with a  $(\hat{W}_j)_{j \in [n]}$  generated conditionally independent of  $(U_j, V_j)_{j \in [n]}$  conditioned on  $(W_j)_{j \in [n]}$ , then only with negligible probability will the test (a) report failure w.r.t.  $V$ , or (b) report success when for more than a small fraction of  $j$ 's,  $\hat{W}_j$  and  $W_j$  map to different values under  $\psi_{W \setminus U^{(r-1)|W}}$  (or  $\psi_{W \setminus U^{(r)}}$  for any  $r$  as  $r = |\mathcal{V}||\mathcal{W}|$  corresponds to the finest partition). The lemma gives similar guarantees when  $(V_j)_{j \in [n]}$  is replaced with a  $(\hat{V}_j)_{j \in [n]}$ .

**Lemma 4.** *Suppose  $(U_j, V_j, W_j)$  are i.i.d. according to  $P_{UVW}$  for all  $j \in [n]$ . For any  $\delta > 0$ , there exist parameters  $\gamma_1, \dots, \gamma_{|\mathcal{V}||\mathcal{W}|} > 0$  such that*

- (i)  $\Sigma_{UVW}$  succeeds with probability  $1 - 2^{-\Omega(n)}$  on input  $(U_j, V_j, W_j)_{j \in [n]}$ .
- (ii) Suppose  $(\hat{W}_j)_{j \in [n]} \longleftrightarrow (W_j)_{j \in [n]} \longleftrightarrow (U_j, V_j)_{j \in [n]}$  is a Markov chain. On input  $(U_j, \hat{V}_j, \hat{W}_j)_{j \in [n]}$ ,
  - (a)  $\Sigma_{UVW}$  reports failure w.r.t.  $V$  with probability  $2^{-\Omega(n)}$ .
  - (b) When  $\ell = |\mathcal{V}||\mathcal{W}|$ ,

$$\Pr \left[ (|\{j : \psi_{W \setminus U^{(\ell)}}(\hat{W}_j) \neq \psi_{W \setminus U^{(\ell)}}(W_j)\}| > n\delta) \wedge (\Sigma_{UVW} \text{ reports success}) \right] \leq 2^{-\Omega(n)}.$$

- (iii) Suppose  $(\hat{V}_j)_{j \in [n]} \longleftrightarrow (V_j)_{j \in [n]} \longleftrightarrow (U_j, W_j)_{j \in [n]}$  is a Markov chain. On input  $(U_j, \hat{V}_j, W_j)_{j \in [n]}$ ,
  - (a)  $\Sigma_{UVW}$  reports failure w.r.t.  $W$  with probability  $2^{-\Omega(n)}$ .
  - (b) When  $\ell = |\mathcal{V}||\mathcal{W}|$ ,

$$\Pr \left[ (|\{j : \psi_{V \setminus U^{(\ell)}}(\hat{V}_j) \neq \psi_{V \setminus U^{(\ell)}}(V_j)\}| > n\delta) \wedge (\Sigma_{UVW} \text{ reports success}) \right] \leq 2^{-\Omega(n)}.$$

See [30, Appendix C.1] for a proof which makes repeated uses of Lemma 1.

We will show that the following protocol implements pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  using the correlation  $(X, Y, Z)$ .

Parties  $P_1, P_2$  and  $P_3$  receive inputs  $(X_{i,j})_{i \in [m], j \in [n]}$ ,  $(Y_{i,j})_{i \in [m], j \in [n]}$ , and  $(Z_{i,j})_{i \in [m], j \in [n]}$ , respectively, such that, for all  $i \in [m], j \in [n]$ ,  $(X_{i,j}, Y_{i,j}, Z_{i,j})$

are i.i.d. according to  $P_{XYZ}$ . The parameters  $m, n$ , and  $\gamma > 0$  in the protocol will be specified during the security analysis.

**Signing Phase**

1.  $P_3$  uniformly samples  $S \subset [m]$  conditioned on  $|S| = m/2$  and sends  $S, (\hat{Z}_{i,j})_{i \in S, j \in [n]} = (Z_{i,j})_{i \in S, j \in [n]}$  to  $P_2$ .
2. To send message  $b \in \{0, 1\}$ ,  $P_1$  sends  $b$  and  $(\tilde{X}_{i,j})_{i \in [m], j \in T_b} = (X_{i,j})_{i \in [m], j \in T_b}$  to  $P_2$ , where  $T_b = \{i : \frac{bn}{2} + 1 \leq i \leq \frac{(b+1)n}{2}\}$ .
3.  $P_2$  accepts  $b$  unless for some  $i \in S$  ( $S$  of size  $m/2$ ) the statistical test  $\Sigma_{YZX}$  with input  $(Y_{i,j}, \hat{Z}_{i,j}, \tilde{X}_{i,j})_{j \in T_b}$  fails w.r.t.  $X$ . In the latter case  $P_2$  rejects  $b$ .

**Transfer Phase**

4. To transfer a message  $b$  it accepted,  $P_2$  sends  $\hat{b} = b$  and  $(\check{X}_{i,j}, \check{Y}_{i,j})_{i \in [m], j \in T_{\hat{b}}} = (\tilde{X}_{i,j}, Y_{i,j})_{i \in [m], j \in T_{\hat{b}}}$  to  $P_3$ .
5.  $P_1$  sends  $(\hat{X}_{i,j})_{i \in [m], j \in [n]} = (X_{i,j})_{i \in [m], j \in [n]}$  to  $P_3$ .
6. For each  $i \in [m] \setminus S$ ,  $P_3$  runs  $\Sigma_{ZXY}$  with input  $(Z_{i,j}, \hat{X}_{i,j}, \check{Y}_{i,j})_{j \in T_{\hat{b}}}$ . If for some  $i \in [m] \setminus S$ ,  $\Sigma_{ZXY}$  fails w.r.t.  $X$ ,  $P_3$  accepts  $\hat{b}$ , else if  $\Sigma_{ZXY}$  fails w.r.t.  $Y$  for some  $i \in [m] \setminus S$ ,  $P_3$  rejects. If  $\Sigma_{ZXY}$  reports success for all  $i \in [m] \setminus S$ , go to the next step.
7. Denote  $\psi_{X \setminus Y(|\mathcal{X}| \geq 1)}$  by  $f^\dagger$  and  $\psi_{X \setminus Z(|\mathcal{X}| \geq 1)}$  by  $f^\ddagger$ .  $P_3$  rejects  $\hat{b}$  if there are more than  $m/4$  distinct  $i \in [m] \setminus S$  such that

$$(f^\dagger(\check{X}_{i,j}), Z_{i,j}, f^\ddagger(\hat{X}_{i,j}))_{j \in T_{\hat{b}}} \notin \mathcal{T}_\gamma^{n/2}(P_{X^\dagger Z X^\ddagger}).$$

$P_3$  accepts  $\hat{b}$  otherwise.

*Proof (Theorem 5).* Let  $\delta, \gamma > 0$  (to be decided) and set the parameters for the statistical tests  $\Sigma_{YZX}$  and  $\Sigma_{ZXY}$  in the protocol from Lemma 4. We first argue the correctness of the protocol. Lemma 4(i) guarantees that when all parties behave honestly, each invocation of the tests  $\Sigma_{YZX}$  and  $\Sigma_{ZXY}$  in the protocol reports success with probability  $1 - 2^{-\Omega(n)}$ . Furthermore, since  $\gamma > 0$ , each typicality check made by  $P_3$  in step 7 succeeds with probability  $1 - 2^{-\Omega(n)}$  by Theorem 4. Thus, by a union bound,  $P_2$  and  $P_3$  together accept  $P_1$ 's message with probability  $1 - m2^{-\Omega(n)}$ .

We will separately consider the cases where  $P_1, P_2$ , and  $P_3$  are corrupt.

*Security against  $P_3$  (Correctness).* Suppose  $P_3$  sends  $S \subset [m]$  and  $(\hat{Z}_{i,j})_{i \in S, j \in [n]}$  to  $P_2$  in step 1. To prove security against  $P_3$ , it suffices to show that, for all  $i \in S$ , when  $P_2$  runs  $\Sigma_{YZX}$  with input  $(Y_{i,j}, \hat{Z}_{i,j}, X_{i,j})_{j \in T_b}$  in step 3, it reports failure w.r.t.  $X$  with only a negligible probability. Notice that  $(S, (\hat{Z}_{i,j})_{i \in S, j \in [n]})$  satisfies the Markov chain

$$S, (\hat{Z}_{i,j})_{i \in S, j \in [n]} \longleftrightarrow (Z_{i,j})_{i \in [m], j \in [n]} \longleftrightarrow (X_{i,j}, Y_{i,j})_{i \in [m], j \in [n]}. \tag{10}$$



Let us define

$$(\hat{Z}_{i,j})_{i \in [m] \setminus S, j \in T_b} = (Z_{i,j})_{i \in [m] \setminus S, j \in T_b}. \quad (11)$$

Then

$$\begin{aligned} & \Pr \left[ \exists i \in S \text{ s.t. } \Sigma_{YZX} \text{ fails w.r.t. } X \text{ on input } (Y_{i,j}, \hat{Z}_{i,j}, X_{i,j})_{j \in T_b} \right] \\ & \leq \Pr \left[ \exists i \in [m] \text{ s.t. } \Sigma_{YZX} \text{ fails w.r.t. } X \text{ on input } (Y_{i,j}, \hat{Z}_{i,j}, X_{i,j})_{j \in T_b} \right] \\ & \leq \sum_{i=1}^m \Pr \left[ \Sigma_{YZX} \text{ fails w.r.t. } X \text{ on input } (Y_{i,j}, \hat{Z}_{i,j}, X_{i,j})_{j \in T_b} \right], \end{aligned}$$

where the last step is a union bound. To bound each of these probabilities, we notice that by (10)–(11) and the fact that  $(X_{i,j}, Y_{i,j}, Z_{i,j})$  are i.i.d. over  $i \in [m], j \in [n]$ , the following Markov chain holds for each  $i \in [m]$ :

$$(\hat{Z}_{i,j})_{j \in [n]} \longleftrightarrow (Z_{i,j})_{j \in [n]} \longleftrightarrow (X_{i,j}, Y_{i,j})_{j \in [n]} \quad (12)$$

Hence, by Lemma 4(iii)(a), for  $i \in [m]$ ,

$$\Pr \left[ \Sigma_{YZX} \text{ fails w.r.t. } X \text{ on input } (Y_{i,j}, \hat{Z}_{i,j}, X_{i,j})_{j \in T_b} \right] = 2^{-\Omega(|T_b|)} = 2^{-\Omega(n)}.$$

Thus,  $P_2$  accepts  $P_1$ 's message with probability  $1 - m2^{-\Omega(n)}$ .

*Security against  $P_1$  (Transferability).* To prove security against  $P_1$ , it is sufficient to show that, if  $P_2$  accepts, then  $P_3$  also accepts with overwhelming probability. Fix  $b \in \{0, 1\}$ . Suppose  $P_1$  sends  $(b, (\tilde{X}_{i,j})_{i \in [m], j \in T_b})$  to  $P_2$  in step 2 and  $(\hat{X}_{i,j})_{i \in [m], j \in [n]}$  to  $P_3$  in step 5. Then,

$$\begin{aligned} & \left( (\hat{X}_{i,j})_{i \in [m], j \in [n]}, (\tilde{X}_{i,j})_{i \in [m], j \in T_b} \right) \longleftrightarrow (X_{i,j})_{i \in [m], j \in [n]} \\ & \longleftrightarrow \left( S, (Y_{i,j}, Z_{i,j})_{i \in [m], j \in [n]} \right). \end{aligned} \quad (13)$$

Formally, we need to show that the event  $E = (E_1 \vee E_2) \wedge (E_3 \vee (E_4 \wedge E_5))$  occurs with negligible probability, where

1.  $E_1$  is the event “ $\Sigma_{YZX}$  succeeds on input  $(Y_{i,j}, Z_{i,j}, \tilde{X}_{i,j})_{j \in T_b}$  in step 3 for each  $i \in S$ ”.
2.  $E_2$  is the event “ $\Sigma_{YZX}$  fails w.r.t.  $Z$  on input  $(Y_{i,j}, Z_{i,j}, \tilde{X}_{i,j})_{j \in T_b}$  in step 3 for some  $i \in S$ ”.
3.  $E_3$  is the event “ $\Sigma_{ZXY}$  fails w.r.t.  $Y$  on input  $(Z_{i,j}, \hat{X}_{i,j}, Y_{i,j})_{j \in T_b}$  in step 6 for some  $i \in [m] \setminus S$ ”.
4.  $E_4$  is the event “ $\Sigma_{ZXY}$  succeeds on input  $(Z_{i,j}, \hat{X}_{i,j}, Y_{i,j})_{j \in T_b}$  in step 6 for each  $i \in [m] \setminus S$ ”.
5.  $E_5$  is the event “in step 7,  $(f^\dagger(\tilde{X}_{i,j}), Z_{i,j}, f^\ddagger(\hat{X}_{i,j}))_{j \in T_b} \notin \mathcal{T}_\gamma^{\frac{\alpha}{2}}(P_{X^\dagger Z X^\ddagger})$  for at least  $\frac{m}{4}$  distinct values of  $i \in [m] \setminus S$ ”.

Note that, here  $E_1 \vee E_2$  is the event in which  $P_2$  accepts the signature, and  $E_3 \vee (E_4 \wedge E_5)$  is the event in which  $P_3$  rejects the signature. Since  $(E_1 \vee E_2) \wedge (E_3 \vee (E_4 \wedge E_5)) \subset E_2 \vee E_3 \vee (E_1 \wedge E_4 \wedge E_5)$ ,

$$\begin{aligned} \Pr[E] &= \Pr[(E_1 \vee E_2) \wedge (E_3 \vee (E_4 \wedge E_5))] \\ &\leq \Pr[E_2] + \Pr[E_3] + \Pr[E_1 \wedge E_4 \wedge E_5]. \end{aligned} \tag{14}$$

We now bound each of these probabilities. We have

$$\Pr[E_2] = \sum_{\hat{S} \subset [m]: |\hat{S}| = \frac{m}{2}} \Pr[S = \hat{S}] \cdot \Pr[E_2 | S = \hat{S}].$$

Since  $S$  is chosen independent of  $(\tilde{X}_{i,j}, Y_{i,j}, Z_{i,j})_{i \in [m], j \in T_b}$ ,

$$\begin{aligned} \Pr[E_2 | S = \hat{S}] &= \Pr \left[ \exists i \in \hat{S} \text{ s.t. } \Sigma_{YZX} \text{ fails w.r.t. } Z \text{ for } (Y_{i,j}, Z_{i,j}, \tilde{X}_{i,j})_{j \in T_b} \right] \\ &\leq \sum_{i \in \hat{S}} \Pr \left[ \Sigma_{YZX} \text{ fails w.r.t. } Z \text{ on } (Y_{i,j}, Z_{i,j}, \tilde{X}_{i,j})_{j \in T_b} \right] \\ &= |\hat{S}| 2^{-\Omega(|T_b|)} = m 2^{-\Omega(n)}, \end{aligned}$$

where the bound on the probabilities in the last step follows from Lemma 4(ii)(a) since, by (13) and the fact that  $(X_{i,j}, Y_{i,j}, Z_{i,j})$  are i.i.d. over  $i \in [m], j \in [n]$ , the following Markov chain holds for each  $i \in [m]$ :

$$(\tilde{X}_{i,j})_{j \in T_b} \longleftrightarrow (X_{i,j})_{j \in T_b} \longleftrightarrow (Y_{i,j}, Z_{i,j})_{j \in T_b}. \tag{15}$$

Hence,

$$\Pr[E_2] \leq \sum_{\hat{S} \subset [m]: |\hat{S}| = \frac{m}{2}} \Pr[S = \hat{S}] m 2^{-\Omega(n)} = m 2^{-\Omega(n)}. \tag{16}$$

To bound  $\Pr[E_3]$ , using the fact that

$$(\hat{X}_{i,j})_{j \in T_b} \longleftrightarrow (X_{i,j})_{j \in T_b} \longleftrightarrow (Y_{i,j}, Z_{i,j})_{j \in T_b} \tag{17}$$

is a Markov chain for each  $i \in [m]$  and that  $(\hat{X}_{i,j}, Y_{i,j}, Z_{i,j})_{i \in [m], j \in T_b}$  is independent of  $S$ , following similar steps as above (invoking Lemma 4(iii)(a) along the way) we have

$$\begin{aligned} \Pr[E_3] &= \Pr \left[ \exists i \in [m] \setminus S \text{ s.t. } \Sigma_{ZXY} \text{ fails w.r.t. } Y \text{ for } (Z_{i,j}, \hat{X}_{i,j}, Y_{i,j})_{j \in T_b} \right] \\ &= m 2^{-\Omega(n)}. \end{aligned} \tag{18}$$

We now bound  $\Pr[E_1 \wedge E_4 \wedge E_5]$ . Recall that we denote  $f^\dagger = \psi_{X \setminus Y^{(\|\mathcal{X}\| \|\mathcal{Z}\|)}}$  and  $f^\ddagger = \psi_{X \setminus Z^{(\|\mathcal{X}\| \|\mathcal{Y}\|)}}$ . Let us define the events

$$\begin{aligned} B &= \left( \left| \{i \in [m] \setminus S : |\{j \in T_b : f^\dagger(\tilde{X}_{i,j}) \neq f^\dagger(X_{i,j})\}| > n\delta\} \right| \geq \frac{m}{8} \right), \\ C &= \left( \exists i \in [m] \setminus S \text{ such that } |\{j \in T_b : f^\ddagger(\hat{X}_{i,j}) \neq f^\ddagger(X_{i,j})\}| > n\delta \right). \end{aligned}$$

Since  $E_1 \wedge E_4 \wedge E_5 \subset (E_1 \wedge B) \vee (E_4 \wedge C) \vee (E_5 \wedge B^c \wedge C^c)$ ,

$$\Pr[E_1 \wedge E_4 \wedge E_5] \leq \Pr[E_1 \wedge B] + \Pr[E_4 \wedge C] + \Pr[E_5 \wedge B^c \wedge C^c]. \tag{19}$$

We proceed to bound each of these probabilities.

$$\Pr[E_1 \wedge B] \leq \Pr[B \wedge D] + \Pr[E_1 \wedge D^c],$$

where  $D = \bigwedge_{i \in S} (|\{j \in T_b : f^\dagger(\tilde{X}_{i,j}) \neq f^\dagger(X_{i,j})\}| \leq n\delta)$ . For  $i \in [m]$ , if we define  $F_i$  as the indicator random variable of the event  $(|\{j \in T_b : f^\dagger(\tilde{X}_{i,j}) \neq f^\dagger(X_{i,j})\}| > n\delta)$ , then

$$B = \left( \sum_{i=1}^m F_i \geq \frac{m}{8} \right), \quad D = \left( \sum_{i \in S} F_i = 0 \right).$$

Since  $S$  is a random subset of size  $m/2$  uniformly chosen from  $[m]$  independent of  $(\tilde{X}_{i,j}, X_{i,j})_{i \in [m], j \in T_b}$  (and therefore independent of  $(F_i)_{i \in [m]}$ ),

$$\Pr[B \wedge D] = 2^{-\Omega(m)}.$$

Now, to bound  $\Pr[E_1 \wedge D^c]$ , for  $i \in [m]$ , let

$$E_{1,i} = \left( (\Sigma_{YZX} \text{ succeeds for } (Y_{i,j}, Z_{i,j}, \tilde{X}_{i,j})_{j \in T_b}) \wedge (|\{j \in T_b : f^\dagger(\tilde{X}_{i,j}) \neq f^\dagger(X_{i,j})\}| > n\delta) \right).$$

Then

$$\begin{aligned} E_1 \wedge D^c &= \left( (\Sigma_{YZX} \text{ succeeds for } (Y_{i,j}, Z_{i,j}, \tilde{X}_{i,j})_{j \in T_b}, \forall i \in S) \wedge (\exists i \in S \text{ s.t. } |\{j \in T_b : f^\dagger(\tilde{X}_{i,j}) \neq f^\dagger(X_{i,j})\}| > n\delta) \right) \\ &\subset \bigvee_{i \in S} E_{1,i}. \end{aligned}$$

Hence,

$$\begin{aligned} \Pr[E_1 \wedge D^c] &\leq \Pr \left[ \bigvee_{i \in S} E_{1,i} \right] \\ &\leq \sum_{\hat{S} \subset [m]: |\hat{S}| = \frac{m}{2}} \Pr[S = \hat{S}] \sum_{i \in \hat{S}} \Pr[E_{1,i} | S = \hat{S}], \end{aligned}$$

where the last inequality is a union bound. By the independence of  $S$  and  $(\tilde{X}_{i,j}, Y_{i,j}, Z_{i,j})_{i \in [m], j \in T_b}$ ,

$$\begin{aligned} &\Pr[E_{1,i} | S = \hat{S}] \\ &= \Pr \left[ (\Sigma_{YZX} \text{ succeeds for } (Y_{i,j}, Z_{i,j}, \tilde{X}_{i,j})_{j \in T_b}) \wedge (|\{j \in T_b : f^\dagger(\tilde{X}_{i,j}) \neq f^\dagger(X_{i,j})\}| > n\delta) \right] \\ &= 2^{-\Omega(n)}, \end{aligned}$$

where the last step follows from Lemma 4(ii)(b) since the Markov chain (15) holds for each  $i \in [m]$  and  $f^\dagger = \psi_{X \setminus Y^{(\mathcal{X} \parallel \mathcal{Z})}}$ . Thus,  $\Pr[E_1 \wedge B] = 2^{-\Omega(m)} + m2^{-\Omega(n)}$ .

To bound the term  $\Pr[E_4 \wedge C]$  in (19), let us define, for  $i \in [m]$ ,

$$E_{4,i} = \left( (\Sigma_{ZXY} \text{ succeeds for } (Z_{i,j}, \hat{X}_{i,j}, Y_{i,j})_{j \in T_b}) \right. \\ \left. \wedge (|\{j \in T_b : f^\ddagger(\hat{X}_{i,j}) \neq f^\ddagger(X_{i,j})\}| > n\delta) \right).$$

Since  $(E_4 \wedge C) \subset \bigvee_{i \in [m] \setminus S} E_{4,i}$ ,

$$\Pr[E_4 \wedge C] \leq \Pr \left[ \bigvee_{i \in [m] \setminus S} E_{4,i} \right] \\ \leq \sum_{\hat{S} \subset [m]: |\hat{S}| = \frac{m}{2}} \Pr[S = \hat{S}] \sum_{i \in [m] \setminus \hat{S}} \Pr[E_{4,i} | S = \hat{S}].$$

We may bound  $\Pr[E_{4,i} | S = \hat{S}]$  using the Markov chain (17) and the independence of  $(\hat{X}_{i,j}, Y_{i,j}, Z_{i,j})_{i \in [m], j \in T_b}$  and  $S$  following similar steps as in the bound for  $\Pr[E_{1,i} | S = \hat{S}]$  above (now invoking Lemma 4(iii)(b)) to obtain  $\Pr[E_4 \wedge C] \leq m2^{-\Omega(n)}$ .

To bound the term  $\Pr[E_5 \wedge B^c \wedge C^c]$  in (19), we will make use of the following lemma:

**Lemma 5.** *For any distribution  $P_{UV}$  and  $\delta > 0$ , there exists  $\gamma > 0$  that approaches 0 as  $\delta$  approaches 0 such that, for random variables  $(U_j, V_j, \hat{U}_j, \hat{V}_j)_{j \in [n]}$  with  $(U_j, V_j)_{j \in [n]}$  i.i.d. according to  $P_{UV}$ ,*

$$\Pr \left[ (|\{j : \hat{U}_j \neq U_j\}| \leq n\delta) \wedge (|\{j : \hat{V}_j \neq V_j\}| \leq n\delta) \right. \\ \left. \wedge ((\hat{U}_j, \hat{V}_j)_{j \in [n]} \notin \mathcal{T}_\gamma^n(P_{UV})) \right] \leq 2^{-\Omega(n)}. \quad (20)$$

For  $i \in [m]$ , define the events

$$E_{5,i} = \left( ((f^\dagger(\tilde{X}_{i,j}), Z_{i,j}, f^\ddagger(\hat{X}_{i,j}))_{j \in T_b} \notin \mathcal{T}_{\gamma^{\frac{n}{2}}}^n(P_{X^\dagger Z X^\ddagger})) \right. \\ \left. \wedge (|\{j \in T_b : f^\dagger(\tilde{X}_{i,j}) \neq f^\dagger(X_{i,j})\}| \leq n\delta) \right. \\ \left. \wedge (|\{j \in T_b : f^\ddagger(\hat{X}_{i,j}) \neq f^\ddagger(X_{i,j})\}| \leq n\delta) \right).$$

Since  $E_5 \wedge B^c \wedge C^c \subset \bigvee_{i \in [m] \setminus S} E_{5,i}$ ,

$$\Pr[E_5 \wedge B^c \wedge C^c] \leq \Pr \left[ \bigvee_{i \in [m] \setminus S} E_{5,i} \right] \\ \leq \sum_{\hat{S} \subset [m]: |\hat{S}| = \frac{m}{2}} \Pr[S = \hat{S}] \sum_{i \in [m] \setminus \hat{S}} \Pr[E_{5,i} | S = \hat{S}].$$

Once again, since  $(\tilde{X}_{i,j}, \hat{X}_{i,j}, Y_{i,j}, Z_{i,j})_{i \in [m], j \in T_b}$  is independent of  $S$ ,

$$\begin{aligned} & \Pr[E_{5,i}|S = \hat{S}] \\ & \leq \Pr \left[ ((f^\dagger(\tilde{X}_{i,j}), Z_{i,j}, f^\ddagger(\hat{X}_{i,j}))_{j \in T_b} \notin \mathcal{T}_{\gamma^{\frac{m}{2}}}(P_{X^\dagger Z X^\ddagger})) \right. \\ & \quad \wedge (|\{j \in [m] : f^\ddagger(\hat{X}_{i,j}) \neq f^\ddagger(X_{i,j})\}| \leq n\delta) \\ & \quad \left. \wedge (|\{j \in T_b : f^\dagger(\tilde{X}_{i,j}) \neq f^\dagger(X_{i,j})\}| \leq n\delta) \right] \\ & = 2^{-\Omega(n)}, \end{aligned}$$

where the bound in the last step follows from Lemma 5 (take  $P_{UV}$  as  $P_{X^\dagger(ZX^\ddagger)}$ , i.e.,  $U = X^\dagger$  and  $V = (Z, X^\ddagger)$ ) as long as  $\delta > 0$  is sufficiently small for a given choice of  $\gamma > 0$  (we ensure that this is the case at the end of this proof). Hence,  $\Pr[E_5 \wedge B^c \wedge C^c] = m2^{-\Omega(n)}$ . Gathering the bounds for all terms in (19), we have shown that  $\Pr[E_1 \wedge E_4 \wedge E_5] = m2^{-\Omega(n)} + 2^{-\Omega(m)}$ . Together with (14),(16),(18), this proves security against a corrupt  $P_1$  if we choose  $m = n$  (as we will do at the end of the proof).

*Security against  $P_2$  (Unforgeability).* To prove security against  $P_2$ , it is sufficient to show that, when  $P_1$ 's message is  $b$  and  $P_2$  claims in step 4 that it received  $1 - b$  from  $P_1$ , party  $P_3$  rejects with overwhelming probability. Suppose  $P_2$  sends  $(1 - b, (\tilde{X}_{i,j}, \check{Y}_{i,j})_{i \in [m], j \in T_{1-b}})$  to  $P_3$  in step 4. Then, these random variables satisfy the following Markov chain

$$\begin{aligned} & (\tilde{X}_{i,j}, \check{Y}_{i,j})_{i \in [m], j \in T_{1-b}} \\ & \longleftrightarrow (S, (Y_{i,j})_{i \in [m], j \in [n]}, (X_{i,j})_{i \in [m], j \in T_b}, (Z_{i,j})_{i \in S, j \in [n]}) \\ & \longleftrightarrow ((X_{i,j})_{i \in [m], j \in T_{1-b}}, (Z_{i,j})_{i \in [m] \setminus S, j \in [n]}). \end{aligned} \tag{21}$$

$P_3$  accepts  $1 - b$  from  $P_2$  only if event  $E' = E'_1 \vee E'_2$  occurs, where

1.  $E'_1$  is the event “ $\Sigma_{ZXY}$  fails w.r.t.  $X$  on input  $(Z_{i,j}, X_{i,j}, \check{Y}_{i,j})_{j \in T_{1-b}}$  in step 6 for some  $i \in [m] \setminus S$ ”.
2.  $E'_2$  is the event “ $(f^\dagger(\tilde{X}_{i,j}), Z_{i,j}, f^\ddagger(X_{i,j}))_{j \in T_{1-b}}$  is a  $\gamma$ -typical sequence w.r.t.  $P_{X^\dagger Z X^\ddagger}$  for at least  $\frac{m}{4}$  instances of  $i \in [m] \setminus S$ ”.

$$\Pr[E'] = \Pr[E'_1 \vee E'_2] \leq \Pr[E'_1] + \Pr[E'_2]. \tag{22}$$

We will show that these events occur with negligible probability. For  $i \in [m]$ , let

$$E'_{1,i} = (\Sigma_{ZXY} \text{ fails w.r.t. } X \text{ for } (Z_{i,j}, X_{i,j}, \check{Y}_{i,j})_{j \in T_{1-b}}).$$

Then

$$\begin{aligned} \Pr[E'_1] &= \Pr \left[ \bigvee_{i \in [m] \setminus S} E'_{1,i} \right] \\ &\leq \sum_{\hat{S} \subset [m]: |\hat{S}| = \frac{m}{2}} \Pr[S = \hat{S}] \sum_{i \in [m] \setminus \hat{S}} \Pr[E'_{1,i}|S = \hat{S}]. \end{aligned}$$

And for each  $\hat{S} \subset [m]$  of size  $m/2$  and  $i \in [m] \setminus \hat{S}$ ,

$$\begin{aligned} \Pr[E'_{1,i} | S = \hat{S}] &= \Pr \left[ \Sigma_{ZXY} \text{ fails w.r.t. } X \text{ for } (Z_{i,j}, X_{i,j}, \check{Y}_{i,j})_{j \in T_{1-b}} \mid S = \hat{S} \right] \\ &= 2^{-\Omega(|T_b|)} = 2^{-\Omega(n)}, \end{aligned}$$

where the last step follows from Lemma 4(ii)(a) since, conditioned on  $S = \hat{S}$ ,

- (i)  $(X_{i,j}, Y_{i,j}, Z_{i,j})_{i \in [m], j \in [n]}$  is distributed  $P_{XYZ}$  i.i.d. (since  $S$  is independent of  $(X_{i,j}, Y_{i,j}, Z_{i,j})_{i \in [m], j \in [n]}$ ), and
- (ii) by (21) and (i) above, for  $i \in [m] \setminus \hat{S}$ ,

$$(\check{Y}_{i,j})_{j \in T_{1-b}} \longleftrightarrow (Y_{i,j})_{j \in T_{1-b}} \longleftrightarrow (X_{i,j}, Z_{i,j})_{j \in T_{1-b}} \tag{23}$$

is a Markov chain.

Hence,

$$\Pr[E'_1] = m2^{-\Omega(n)}. \tag{24}$$

Finally, to bound  $\Pr[E'_2]$ , we need the following lemma:

**Lemma 6.** *Suppose the Markov chain  $X^\dagger \longleftrightarrow Y \longleftrightarrow (Z, X^\ddagger)$  does not hold for the joint distribution  $P_{XYZ}$ . Let  $(X_j, Y_j, Z_j)_{j \in [n]}$  be i.i.d. according to  $P_{XYZ}$ . For any  $(\hat{X}_j)_{j \in [n]}$  satisfying the Markov chain*

$$(\hat{X}_j)_{j \in [n]} \longleftrightarrow (Y_j)_{j \in [n]} \longleftrightarrow (Z_j, X_j)_{j \in [n]},$$

and all sufficiently small  $\gamma > 0$ ,

$$\Pr \left[ \left( f^\dagger(\hat{X}_j), Z_j, f^\ddagger(X_j) \right)_{j \in [n]} \in \mathcal{T}_\gamma^n(P_{X^\dagger Z X^\ddagger}) \right] \leq 2^{-\Omega(n)}, \tag{25}$$

where  $f^\dagger = \psi_{X \setminus Y(|\mathcal{X}||\mathcal{Z}|)}$  and  $f^\ddagger = \psi_{X \setminus Z(|\mathcal{X}||\mathcal{Y}|)}$ .

For  $i \in [m]$ , let

$$E'_{2,i} = ((f^\dagger(\check{X}_{i,j}), Z_{i,j}, f^\ddagger(X_{i,j}))_{j \in T_{1-b}} \in \mathcal{T}_\gamma^{n/2}(P_{X^\dagger Z X^\ddagger})).$$

Clearly,  $E'_2 \subset \bigvee_{i \in [m] \setminus S} E'_{2,i}$  (in fact,  $E'_2$  requires at least  $m/4$  instances of  $E'_{2,i}$ 's to occur for  $i \in [m] \setminus S$ ). Hence,

$$\begin{aligned} \Pr[E'_2] &= \Pr \left[ \bigvee_{i \in [m] \setminus S} E'_{2,i} \right] \\ &\leq \sum_{\hat{S} \subset [m]: |\hat{S}| = \frac{m}{2}} \Pr[S = \hat{S}] \sum_{i \in [m] \setminus \hat{S}} \Pr[E'_{2,i} | S = \hat{S}]. \end{aligned}$$

As seen in the analysis of  $\Pr[E'_1]$  above, conditioned on  $S = \hat{S}$ , the random variables  $(X_{i,j}, Y_{i,j}, Z_{i,j})_{i \in [m], j \in [n]}$  are distributed  $P_{XYZ}$  i.i.d. Together with (21), this implies that, conditioned on  $S = \hat{S}$ , for all  $i \in [m] \setminus \hat{S}$ ,

$$(\check{Y}_{i,j})_{j \in T_{1-b}} \longleftrightarrow (Y_{i,j})_{j \in T_{1-b}} \longleftrightarrow (X_{i,j}, Z_{i,j})_{j \in T_{1-b}} \tag{26}$$

is a Markov chain. Further, by the hypothesis of the theorem,  $P_{XYZ}$  is such that  $X^\dagger \longleftrightarrow Y \longleftrightarrow (Z, X^\ddagger)$  is not a Markov chain. Hence, for sufficiently small  $\gamma > 0$ , by Lemma 6,  $\Pr[E'_{2,i} | S = \hat{S}] \leq 2^{-\Omega(n)}$  for all  $i \in [m] \setminus \hat{S}$ . Thus,  $\Pr[E'_2] = m2^{-\Omega(n)}$  which together with (22) and (24) gives  $\Pr[E'] = m2^{-\Omega(n)}$ . The proof of security against  $P_2$  follows.

At different points in the proof we made the following assumptions about the parameters  $\gamma > 0, \delta > 0$ : the parameter  $\gamma > 0$  should be sufficiently small as required by Lemma 6 in the proof of security against  $P_2$ , and, for a given  $\gamma > 0$ , the parameter  $\delta > 0$  should be sufficiently small as required by Lemma 5 in the proof of security against  $P_1$ . Clearly, we may simultaneously choose these parameters to satisfy these assumptions. Further, we set  $m = n$  so that the number of samples of the correlation used is  $N = nm = n^2$  and the security parameter, as calculated above, is  $\epsilon = 2^{-\Omega(n)} = 2^{-\Omega(\sqrt{N})}$ , i.e.,  $N = O(\log^2(\frac{1}{\epsilon}))$  as desired.

### 3.2 Altering the Transfer Path of a Pseudo-signature Protocol

**Theorem 6.** *A pseudo-signature protocol with transfer path  $P_1 \rightarrow P_3 \rightarrow P_2$  implies the existence of a pseudo-signature protocol with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$ .*

*Proof.* Let  $\Pi$  be an  $\epsilon$ -secure pseudo-signature protocol for the transfer path  $P_1 \rightarrow P_3 \rightarrow P_2$ . We build an  $\eta$ -secure pseudo-signature protocol with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$ , where  $\eta = O(\epsilon \log(\frac{1}{\epsilon}))$ . Fix a number  $n = \Theta(\log(\frac{1}{\epsilon}))$ . This protocol makes  $n$  independent invocations  $\Pi_i, i \in [n]$  of the given protocol.

**Setup phase** (*Establishing a new correlation*).

1.  $P_1$  samples uniformly random bits  $U_1, \dots, U_n$ . For  $i \in [n]$ ,  $P_1$  signs  $U_i$  and sends to  $P_3$  using the *signing phase* of  $\Pi_i$ .
2. If  $P_3$  rejects any of these  $n$  messages at the end of the respective signature phases, it aborts the *setup phase* after informing  $P_2$ .
3.  $P_2$  picks  $S_0 \subset [n/2], S_1 \subset [n] \setminus [n/2]$  with  $|S_0| = |S_1| = n/4$  uniformly at random and sends  $S_0 \cup S_1$  to  $P_3$ .
4. If  $P_3$  has not aborted the *setup phase* in step 2, for each  $i \in S_0 \cup S_1$ ,  $P_3$  sends  $\tilde{U}_i = U_i$  to  $P_2$  using the *transfer phase* of  $\Pi_i$ .
5. If for any  $i \in S_0 \cup S_1$ ,  $P_2$  rejects  $\tilde{U}_i$  at the end of *transfer phase* of  $\Pi_i$ , then  $P_2$  flags  $P_3$  as corrupt.

**Signing phase**  $P_1 \rightarrow P_2$ .

6. To send message  $b \in \{0, 1\}$ ,  $P_1$  sends  $b$  and  $(\hat{U}_i)_{i \in T_b} = (U_i)_{i \in T_b}$  to  $P_2$  where  $T_b = \{i : \frac{bn}{2} + 1 \leq i \leq \frac{(b+1)n}{2}\}$ .

7. If  $P_2$  flagged  $P_3$  as corrupt in step 5, or  $P_2$  was informed by  $P_3$  in step 2 that it is aborting the *setup phase*, then  $P_2$  accepts  $b$ . Else,  $P_2$  accepts if  $\hat{U}_i = \check{U}_i$  for each  $i \in S_b$ , and rejects otherwise.

**Transfer phase**  $P_2 \rightarrow P_3$ .

8. To transfer an accepted  $b$ ,  $P_2$  sends  $\left(\hat{b}, (\check{U}_i)_{i \in T_{\hat{b}}}\right) = \left(b, (\hat{U}_i)_{i \in T_b}\right)$  to  $P_3$ .
9.  $P_3$  accepts  $\hat{b}$  if it aborted the setup phase in step 2. Else, if  $\check{U}_i = U_i$  for at least  $n/6$  distinct  $i \in T_{\hat{b}} \setminus S_{\hat{b}}$ ,  $P_3$  accepts  $\hat{b}$ , and rejects otherwise.

*Security against  $P_1$  (Transferability).* We will show that if  $P_2$  accepts  $b$  in the *signing phase*, then  $P_3$  rejects  $b$  in the *transfer phase* with a negligible probability. Examining steps 7 and 9, this event happens only if (i)  $P_2$  flagged honest  $P_3$  as corrupt in step 5 (in this case  $P_2$  accepts any signature that  $P_1$  sends in the *signing phase*), or (ii)  $(U_i)_{i \in T_b}$  and  $(\hat{U}_i)_{i \in T_b}$  sent by  $P_1$  in steps 1 and 6, respectively, are such that  $\hat{U}_i = U_i$  for all  $i \in S_b$  and  $\hat{U}_i \neq U_i$  for more than  $n/4 - n/6 = n/12$  distinct  $i \in T_b \setminus S_b$ . By  $\varepsilon$ -security of  $\Pi$ , if  $P_3$  accepts a message in the signing phase of  $\Pi$ , then  $P_2$  rejects it in the transfer phase with probability  $\varepsilon$ . Hence, by a union bound, (i) occurs with at most  $n\varepsilon/2$  probability. Since  $S_b$  is chosen uniformly at random independent of  $(U_i, \hat{U}_i)_{i \in T_b}$  (as  $S_b$  is unknown to  $P_1$ ), by Chernoff's bound, (ii) occurs with probability  $2^{-\Omega(n)}$ .

*Security against  $P_2$  (Unforgeability).* Suppose  $P_1$ 's message is  $b$ , but in the *transfer phase*,  $P_2$  sends  $(1 - b, (\check{U}_i)_{i \in T_{1-b}})$  to  $P_3$ . Examining step 9,  $P_3$  accepts  $1 - b$  only if (i)  $P_3$  aborted the *setup phase* in step 2, or (ii)  $\check{U}_i = U_i$  for at least  $n/6$  distinct  $i \in T_{1-b} \setminus S_{1-b}$ . By  $\varepsilon$ -security of  $\Pi$ , if  $P_1$  is honest, then  $P_3$  rejects the message in the signing phase with probability  $\varepsilon$ . Hence, by a union bound, (i) occurs with at most  $n\varepsilon/2$  probability. Each  $U_i, i \in T_{1-b} \setminus S_{1-b}$  is chosen uniformly and independently from  $\{0, 1\}$  (unknown to  $P_2$ ), hence  $\Pr[\check{U}_i = U_i] = 1/2$  for  $i \in T_{1-b} \setminus S_{1-b}$ . Hence, by Chernoff's bound, (ii) occurs with probability  $2^{-\Omega(n)}$ .

*Security against  $P_3$  (Correctness).* This amounts to showing that  $P_2$  rejects the message from  $P_1$  in the signing phase with a negligible probability. Examining step 7, this occurs only if  $P_2$  has not flagged  $P_3$  as corrupt in step 5,  $P_3$  did not report abort in step 2, and  $\hat{U}_i \neq \check{U}_i$  for some  $i \in S_b$ . Noting that  $\hat{U}_i = U_i$  for all  $i \in [n]$ , this occurs only if, for some  $i \in S_b$ ,  $P_2$  accepts  $\check{U}_i = 1 - U_i$  at the end of the transfer phase of  $\Pi_i$  when  $P_1$ 's input is  $U_i$ . By  $\varepsilon$ -security of  $\Pi$ , for any  $i \in [n]$ , this occurs with probability  $\varepsilon$ . Hence, we may bound the probability of this event (over any arbitrary choice of  $S_b$ ) by  $n\varepsilon/2$ .

For  $n = \Theta(\log \frac{1}{\varepsilon})$ , these error probabilities compound to  $O(\varepsilon \log(\frac{1}{\varepsilon}))$ . Hence, the above protocol is  $\eta$ -secure.

### 4 Impossibility

**Theorem 7.** *Let  $(X_1, Y_1, Z_1), (X_2, Y_2, Z_2), \dots, (X_n, Y_n, Z_n)$  be independent and identically distributed (i.i.d.) triples with distribution  $P_{XYZ}$ . Suppose parties*



$P_1, P_2, P_3$  have access to correlations  $(X_i)_{i \in [n]}, (Y_i)_{i \in [n]}, (Z_i)_{i \in [n]}$ , respectively, and are connected pairwise by secure channels.

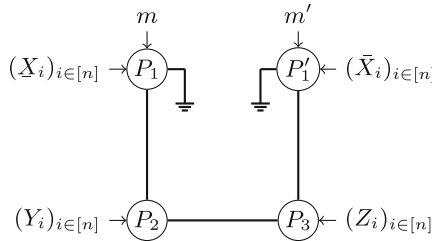
For any  $n$ , and  $\epsilon < 1/3$ , there is no  $\epsilon$ -secure broadcast protocol for the three parties with sender  $P_1$  if the following Markov chain holds:<sup>7</sup>

$$X^\dagger \longleftrightarrow Y \longleftrightarrow Z \longleftrightarrow X^\ddagger. \tag{27}$$

We prove Theorem 7 in two steps. We first generalize the technique of Fischer, Lynch, and Merritt [13] to our setting with correlations. As in [13], we consider a new system with copies of the parties (in fact, only making two copies of party  $P_1$  suffices) and the connections rewired (see Fig. 1). The key step is in identifying a judiciously chosen correlation  $(Q_{\underline{X}Y Z \bar{X}}$  in the lemma below) to feed the parties in the new network. Lemma 7 below gives the conditions on this correlation  $Q_{\underline{X}Y Z \bar{X}}$  under which an impossibility can be shown. We then establish Theorem 7 by showing that such a  $Q_{\underline{X}Y Z \bar{X}}$  exists when (27) holds.

**Lemma 7.** Consider the setup of Theorem 7. For any  $n, \epsilon < 1/3$ , there is no  $\epsilon$ -secure broadcast protocol for the three parties with sender  $P_1$  if there is a distribution  $Q_{\underline{X}Y Z \bar{X}}$  such that  $Q_{\underline{X}Y} = P_{XY}, Q_{YZ} = P_{YZ}, Q_{Z\bar{X}} = P_{ZX}$ , and

- (i)  $\exists Q_{\tilde{Z}|\underline{X}Y Z \bar{X}}$  s.t.  $Q_{\tilde{Z}|\underline{X}Y} = P_{Z|XY}$  and  $(\underline{X}, Y) \longleftrightarrow \tilde{Z} \longleftrightarrow (Z, \bar{X})$ ,
- (ii)  $\exists Q_{\tilde{Y}|\underline{X}Y Z \bar{X}}$  s.t.  $Q_{\tilde{Y}|Z\bar{X}} = P_{Y|ZX}$  and  $(Z, \bar{X}) \longleftrightarrow \tilde{Y} \longleftrightarrow (\underline{X}, Y)$ ,
- (iii)  $\exists Q_{\tilde{X}|\underline{X}Y Z \bar{X}}$  s.t.  $Q_{\tilde{X}|YZ} = P_{X|YZ}$  and  $(Y, Z) \longleftrightarrow \tilde{X} \longleftrightarrow (\underline{X}, \bar{X})$ .



**Fig. 1.** A wiring diagram. Parties  $P_1, P_2, P_3$  and  $P'_1$  observe correlations  $(\underline{X}_i)_{i \in [n]}, (Y_i)_{i \in [n]}, (Z_i)_{i \in [n]}$ , and  $(\bar{X}_i)_{i \in [n]}$ , respectively, i.i.d. according to  $Q_{\underline{X}Y Z \bar{X}}$ . Party  $P_1$  with input  $m$ , and  $P'_1$  with input  $m' \neq m$  are identical copies of the sender. All parties run the protocol honestly.

*Proof (Lemma 7).* Suppose there is a (for now, a perfectly secure) broadcast protocol with sender  $P_1$ . Consider the wiring diagram<sup>8</sup> in Fig. 1 where the correlations  $(\underline{X}_i)_{i \in [n]}, (Y_i)_{i \in [n]}, (Z_i)_{i \in [n]}, (\bar{X}_i)_{i \in [n]}$  of the parties  $P_1, P_2, P_3, P'_1$ , respectively, are i.i.d. according to  $Q_{\underline{X}Y Z \bar{X}}$ . Note that  $P_1$  and  $P'_1$  are identical copies of the sender with the only difference being the correlated observations they use ( $(\underline{X}_i)_{i \in [n]}$  for  $P_1$  and  $(\bar{X}_i)_{i \in [n]}$  for  $P'_1$ ) and their messages ( $m$  for  $P_1$  and  $m' \neq m$  for  $P'_1$ ).

<sup>7</sup> We write “ $T \longleftrightarrow U \longleftrightarrow V \longleftrightarrow W$ ” to mean  $P_{TUVW} = P_T P_{U|T} P_{V|U} P_{W|V}$ .

<sup>8</sup> The wiring diagram is similar to the one described in [19, proof of Theorem 7], but the rest of the argument (including the identification of a correlation  $Q_{\underline{X}Y Z \bar{X}}$  to establish an impossibility) is different.

- $P_1$  is connected to  $P_2$ , but it is disconnected from  $P_3$ . Its messages to  $P_3$  are lost and it receives no messages from  $P_3$ .
- $P_2$  is connected to  $P_1$  and  $P_3$ .
- $P_3$  is connected to  $P_2$  and  $P'_1$  (instead of  $P_1$ ).
- $P'_1$  is connected to  $P_3$  and disconnected from  $P_2$ .

Note that all parties here are honest. We will give three different interpretations of the new system in the diagram as instantiations of the original system (where the correlations are  $P_{XYZ}$  i.i.d.). In each interpretation, two of the parties are honest and the third party, who is dishonest, simulates the remaining two nodes in the new system. This will lead us to a contradiction establishing the impossibility of broadcast.

*First interpretation: malicious  $P_3$ .* Since  $Q_{XY} = P_{XY}$ , the joint distribution of the correlated observations  $(\underline{X}_i)_{i \in [n]}$  and  $(Y_i)_{i \in [n]}$  of  $P_1$  and  $P_2$  are as in the original system. We will argue that the joint view of  $P_1$  and  $P_2$  in the new system is identical to that in the original system where  $P_1$  (with input  $m$ ) and  $P_2$  are honest and  $P_3$  is corrupt as follows: corrupt  $P_3$  in the original system simulates  $P_3$  and  $P'_1$  of the new system, i.e., the corrupt  $P_3$  ignores messages from  $P_1$ , does not send any messages to  $P_1$ , and simulates  $P_3$  and  $P'_1$  by sending/receiving messages on the communication link with  $P_2$ . In order to simulate  $P_3$  and  $P'_1$  of the new system, the corrupt  $P_3$  must be able to generate the observations of these nodes (i.e.,  $(Z_i, \bar{X}_i)_{i \in [n]}$ ) in such a way that they are jointly distributed with  $(\underline{X}_i, Y_i)_{i \in [n]}$  of  $P_1$  and  $P_2$  i.i.d. according to the distribution  $Q_{\underline{X}YZ\bar{X}}$ . For this, the corrupt  $P_3$  may make use of the  $Z$ -observations it receives which are jointly distributed with the observations  $(\underline{X}_i, Y_i)_{i \in [n]}$  of  $P_1$  and  $P_2$  i.i.d. according to  $P_{XYZ}$ . Since (by the lemma's hypothesis (i))  $Q_{\underline{X}Y}Q_{\bar{Z}|\underline{X}Y} = P_{XY}P_{Z|XY} = P_{XYZ}$ , we may treat the observations received by  $P_3$  as  $(\bar{Z}_i)_{i \in [n]}$ . Then, by virtue of the Markov chain  $(\underline{X}, Y) \longleftrightarrow \bar{Z} \longleftrightarrow (Z, \bar{X})$  (of lemma's hypothesis (i)), the corrupt  $P_3$  may sample  $(Z_i, \bar{X}_i)_{i \in [n]}$  to simulate  $P_3$  and  $P'_1$ . Specifically, the conditional independence of  $(\bar{X}, Z)$  and  $(\underline{X}, Y)$  conditioned on  $\bar{Z}$  means that the corrupt  $P_3$  may locally generate the samples of  $\bar{X}$  and  $Z$  (which it needs to simulate  $P'_1$  and  $P_3$ ) using only the samples of  $\bar{Z}$  it observes (and without knowing the samples of  $\underline{X}$  and  $Y$  of  $P_1$  and  $P_2$  which it does not have access to). Thus, the joint view of  $P_1$  and  $P_2$  in the new system is identical to that in the original system where  $P_1$  (with input  $m$ ) and  $P_2$  are honest and  $P_3$  is corrupt. Therefore,  $P_2$  in the new system must output  $m$ .

*Second interpretation: malicious  $P_2$ .* Arguing similarly using  $Q_{Z\bar{X}} = P_{ZX}$  and hypothesis (ii) of the lemma, we may conclude that  $P_3$  in the new system must output  $m'$ .

*Third interpretation: malicious  $P_1$ .* Similarly, using  $Q_{YZ} = P_{YZ}$  and hypothesis (iii), we may conclude that the joint view of  $P_2$  and  $P_3$  in the new system is identical to that in the original system where  $P_2$  and  $P_3$  are honest and  $P_1$  is corrupt. Hence the outputs of  $P_2$  and  $P_3$  must agree, a contradiction.

The above discussion assumed a perfectly secure broadcast. For an  $\epsilon$ -secure broadcast, the conclusions in each interpretation are guaranteed to hold with probability  $1 - \epsilon$ . So, we arrive at contradiction if  $\epsilon < 1/3$ .

Theorem 7 now follows from the lemma below which is proved in [30].

**Lemma 8.** *If  $P_{XYZ}$  is such that the Markov chain of (27) holds, the distribution  $Q_{\underline{X}Yz\bar{X}}$  defined below satisfies  $Q_{\underline{X}Y} = P_{XY}$ ,  $Q_{YZ} = P_{YZ}$ ,  $Q_{Z\bar{X}} = P_{ZX}$ , and conditions (i)-(iii) in the hypothesis of Lemma 7.*

$$Q_{\underline{X}Yz\bar{X}}(\underline{X}, y, z, \bar{X}) \stackrel{\text{def}}{=} P_{YZ}(y, z)P_{X|Y}(\underline{X}|y)P_{X|Z}(\bar{X}|z), \quad \forall \underline{X}, y, z, \bar{X}. \quad (28)$$

## 5 Characterizations

In Theorem 5, we have shown that pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  is feasible from correlation  $(X, Y, Z)$  if the Markov chain  $X^\dagger \longleftrightarrow Y \longleftrightarrow (Z, X^\dagger)$  does not hold; by symmetry, we can show that pseudo-signature with transfer path  $P_1 \rightarrow P_3 \rightarrow P_2$  is feasible from correlation  $(X, Y, Z)$  if the Markov chain  $X^\ddagger \longleftrightarrow Z \longleftrightarrow (Y, X^\ddagger)$  does not hold. Furthermore, in Theorem 6, we have shown that pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  is feasible if pseudo-signature with transfer path  $P_1 \rightarrow P_3 \rightarrow P_2$  is, and vice versa. Thus, we can conclude that pseudo-signatures with both transfer paths  $P_1 \rightarrow P_2 \rightarrow P_3$  and  $P_1 \rightarrow P_3 \rightarrow P_2$  are feasible if at least one of the following Markov chains do not hold.

$$X^\dagger \longleftrightarrow Y \longleftrightarrow (Z, X^\dagger) \quad (29)$$

$$X^\ddagger \longleftrightarrow Z \longleftrightarrow (Y, X^\ddagger) \quad (30)$$

On the other hand, in Theorem 7, we have shown that broadcast with sender  $P_1$  is infeasible if the following Markov chain holds.

$$X^\dagger \longleftrightarrow Y \longleftrightarrow Z \longleftrightarrow X^\ddagger. \quad (31)$$

In fact, the Markov chain condition (31) and the pair of Markov chain conditions (29) and (30) are equivalent: clearly (31) implies (29) and (30); to verify the opposite implication, note that (29) implies that

$$P_{X^\dagger Y Z X^\ddagger} = P_{X^\dagger} P_{Y|X^\dagger} P_{Z X^\ddagger|Y} = P_{X^\dagger} P_{Y|X^\dagger} P_{Z|Y} P_{X^\ddagger|ZY}.$$

Now, (30) implies that  $X^\ddagger \longleftrightarrow Z \longleftrightarrow Y$ , i.e.,  $P_{X^\ddagger|ZY} = P_{X^\ddagger|Z}$ . Hence, (29) and (30) together imply that  $P_{X^\dagger Y Z X^\ddagger} = P_{X^\dagger} P_{Y|X^\dagger} P_{Z|Y} P_{X^\ddagger|Z}$ , which is (31). Using this observation, we can provide a complete characterization of feasibility of pseudo-signature and broadcast. Even though the conditions on the correlation are the same for pseudo-signature and broadcast, we state them separately to clarify the logical difference of the derivations.

**Theorem 8 (Characterization of pseudo-signature).** *Pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  is feasible from correlation  $(X, Y, Z)$  if and only if the Markov chain (31) does not hold. The same statement holds for pseudo-signature with transfer path  $P_1 \rightarrow P_3 \rightarrow P_2$ .*

*Proof.* The “if” part follows from Theorem 5 and Theorem 6. On the other hand, the ‘only if’ part follows from the fact that pseudo-signature with either transfer implies broadcast with sender  $P_1$  and Theorem 7.

**Theorem 9 (Characterization of broadcast).** *Broadcast with sender  $P_1$  is feasible from correlation  $(X, Y, Z)$  if and only if the Markov chain (31) does not hold.*

*Proof.* The “if” part follows from Theorem 5 (invoked for either transfer path with signer  $P_1$ ) and the fact that the pseudo-signature with signer  $P_1$  implies broadcast with sender  $P_1$ . On the other hand, the “only if” part follows from Theorem 7.

## 6 Characterizations for Pseudo-signatures with Limited Connectivity

Notice that the construction in Sect. 3.1 only used the unidirectional links  $P_1 \Rightarrow P_2$ ,  $P_1 \Rightarrow P_3$ , and the bidirectional link  $P_2 \Leftrightarrow P_3$ , while the impossibility in Sect. 4 (via the fact that pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  implies broadcast with sender  $P_1$ ) applies for pseudo-signature protocols which may use all pairwise links in either direction. In this section we consider networks with more limited connectivity than what the construction in Sect. 3.1 demands and give characterizations. Note that a similar question is uninteresting for broadcast with sender  $P_1$  as it is easy to argue that broadcast is impossible without all the links demanded by our construction present.

For pseudo-signature protocols with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$ , since the links  $P_1 \Rightarrow P_2$  and  $P_2 \Rightarrow P_3$  are needed, we consider cases where one or both of the links  $P_1 \Rightarrow P_3$  and  $P_3 \Rightarrow P_2$  that our construction additionally needs are absent.

**Pseudo-signatures with Connectivity  $P_1 \Leftrightarrow P_2 \Leftrightarrow P_3$ .** This network does not have the  $P_1 \Rightarrow P_3$  link our construction needs. This prevents  $P_3$  from being upgraded. However, the partial upgrade of  $P_2$  may still be implemented. Straightforward modifications of the construction (to only use  $P_1 \Rightarrow P_2$  and  $P_2 \Leftrightarrow P_3$  links) show that a pseudo-signature protocol with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  is feasible as long as  $X^\dagger \longleftrightarrow Y \longleftrightarrow Z$  does not hold. As we show in [30, Appendix E.1], this turns out to be the characterizing condition.

**Theorem 10.** *Pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  is feasible from correlation  $(X, Y, Z)$  when secure links  $P_1 \Leftrightarrow P_2$  and  $P_2 \Leftrightarrow P_3$  are available if and only if the following Markov chain does not hold*

$$X^\dagger \longleftrightarrow Y \longleftrightarrow Z. \quad (32)$$

**Pseudo-signatures with No  $P_3 \Rightarrow P_2$  Link.** Here the link between  $P_2$  to  $P_3$  is unidirectional. This prevents  $P_2$  from being upgraded, but  $P_3$  may still be upgraded. With straightforward changes to use only  $P_1 \Rightarrow P_2$ ,  $P_1 \Rightarrow P_3$ , and  $P_2 \Rightarrow P_3$  links, our construction gives a pseudo-signature protocol if  $(X \searrow Y) \longleftrightarrow Y \longleftrightarrow (Z, X^\ddagger)$  does not hold. This turns out to be the characterizing condition (see [30, Appendix E.2]).

**Theorem 11.** *Pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  is feasible from correlation  $(X, Y, Z)$  when secure links  $P_1 \Leftrightarrow P_2$ ,  $P_1 \Leftrightarrow P_3$ , and  $P_2 \Rightarrow P_3$  are available if and only if the following Markov chain does not hold*

$$(X \searrow Y) \longleftrightarrow Y \longleftrightarrow (Z, X^\dagger). \quad (33)$$

**Pseudo-signatures with Connectivity  $P_1 \Leftrightarrow P_2 \Rightarrow P_3$ .** In this third case both  $P_1 \Rightarrow P_3$  and  $P_3 \Rightarrow P_2$  links are absent. This prevents both  $P_2$  and  $P_3$  from being upgraded. The pseudo-signature protocol in [19] (or by altering our construction) which uses only the  $P_1 \Rightarrow P_2$  and  $P_2 \Rightarrow P_3$  links is secure as long as  $X \searrow Y \longleftrightarrow Y \longleftrightarrow Z$  is not true. This is also the characterizing condition for this case (see [30, Appendix E.3]).

**Theorem 12.** *Pseudo-signature with transfer path  $P_1 \rightarrow P_2 \rightarrow P_3$  is feasible from correlation  $(X, Y, Z)$  when secure links  $P_1 \Leftrightarrow P_2$  and  $P_2 \Rightarrow P_3$  are available if and only if the following Markov chain does not hold*

$$X \searrow Y \longleftrightarrow Y \longleftrightarrow Z. \quad (34)$$

## References

1. Ahlswede, R., Csiszár, I.: Common randomness in information theory and cryptography-part I: secret sharing. *IEEE Trans. Inform. Theory* **39**(4), 1121–1132 (1993). <https://doi.org/10.1109/18.243431>
2. Ahlswede, R., Csiszár, I.: On oblivious transfer capacity. *Inf. Theory, Comb. Search Theory*, pp. 145–166 (2013). [https://doi.org/10.1007/978-3-642-36899-8\\_6](https://doi.org/10.1007/978-3-642-36899-8_6)
3. Beaver, D.: Multiparty protocols tolerating half faulty processors. In: Brassard, G. (ed.) *CRYPTO 1989*. LNCS, vol. 435, pp. 560–572. Springer, New York (1990). [https://doi.org/10.1007/0-387-34805-0\\_49](https://doi.org/10.1007/0-387-34805-0_49)
4. Chaum, D., Roijakkers, S.: Unconditionally-secure digital signatures. In: Menezes, A.J., Vanstone, S.A. (eds.) *CRYPTO 1990*. LNCS, vol. 537, pp. 206–214. Springer, Heidelberg (1991). [https://doi.org/10.1007/3-540-38424-3\\_15](https://doi.org/10.1007/3-540-38424-3_15)
5. Chen, J., Micali, S.: Algorand: a secure and efficient distributed ledger. *Theoret. Comput. Sci.* **777**, 155–183 (2019). <https://doi.org/10.1016/j.tcs.2019.02.001>
6. Considine, J., Fitz, M., Franklin, M., Levin, L.A., Maurer, U., Metcalf, D.: Byzantine agreement given partial broadcast. *J. Cryptol.* **18**(3), 191–217 (2005). <https://doi.org/10.1007/s00145-005-0308-x>
7. Cramer, R., Damgård, I., Nielsen, J.: *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, Cambridge (2015). <https://doi.org/10.1017/CBO9781107337756>
8. Crépeau, C.: Equivalence between two flavours of oblivious transfers. In: Pomerance, C. (ed.) *CRYPTO 1987*. LNCS, vol. 293, pp. 350–354. Springer, Heidelberg (1988). [https://doi.org/10.1007/3-540-48184-2\\_30](https://doi.org/10.1007/3-540-48184-2_30)
9. Crépeau, C.: Efficient Cryptographic Protocols Based on Noisy Channels. In: Fumy, W. (ed.) *EUROCRYPT 1997*. LNCS, vol. 1233, pp. 306–317. Springer, Heidelberg (1997). [https://doi.org/10.1007/3-540-69053-0\\_21](https://doi.org/10.1007/3-540-69053-0_21)
10. Crépeau, C., Kilian, J.: Weakening security assumptions and oblivious transfer. In: Goldwasser, S. (ed.) *CRYPTO 1988*. LNCS, vol. 403, pp. 2–7. Springer, New York (1990). [https://doi.org/10.1007/0-387-34799-2\\_1](https://doi.org/10.1007/0-387-34799-2_1)

11. Csiszár, I., Körner, J.: *Information Theory: Coding Theorems for Discrete Memoryless Systems*, 2nd edn. Cambridge University Press, Cambridge (2011). <https://doi.org/10.1017/CBO9780511921889>
12. Dowsley, R., Nascimento, A.C.: On the oblivious transfer capacity of generalized erasure channels against malicious adversaries: the case of low erasure probability. *IEEE Trans. Inform. Theory* **63**(10), 6819–6826 (2017). <https://doi.org/10.1109/TIT.2017.2735423>
13. Fischer, M.J., Lynch, N.A., Merritt, M.: Easy impossibility proofs for distributed consensus problems. *Distrib. Comput.* **1**(1), 26–39 (1986). <https://doi.org/10.1007/BF01843568>
14. Fitzi, M., Gisin, N., Maurer, U.: Quantum solution to Byzantine agreement problem. *Phys. Rev. Lett.* **87**(21), 217901 (2001). <https://doi.org/10.1103/PhysRevLett.87.217901>
15. Fitzi, M., Garay, J.A., Maurer, U., Ostrovsky, R.: Minimal complete primitives for secure multi-party computation. In: Kilian, J. (ed.) *CRYPTO 2001*. LNCS, vol. 2139, pp. 80–100. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-44647-8\\_5](https://doi.org/10.1007/3-540-44647-8_5)
16. Fitzi, M., Gisin, N., Maurer, U., von Rotz, O.: Unconditional Byzantine agreement and multi-party computation secure against dishonest minorities from scratch. In: Knudsen, L.R. (ed.) *EUROCRYPT 2002*. LNCS, vol. 2332, pp. 482–501. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-46035-7\\_32](https://doi.org/10.1007/3-540-46035-7_32)
17. Fitzi, M., Hirt, M., Holenstein, T., Wullschlegel, J.: Two-threshold broadcast and detectable multi-party computation. In: Biham, E. (ed.) *EUROCRYPT 2003*. LNCS, vol. 2656, pp. 51–67. Springer, Heidelberg (2003). [https://doi.org/10.1007/3-540-39200-9\\_4](https://doi.org/10.1007/3-540-39200-9_4)
18. Fitzi, M., Maurer, U.M.: From partial consistency to global broadcast. In: *32nd ACM STOC*, pp. 494–503. ACM Press (May 2000). <https://doi.org/10.1145/335305.335363>
19. Fitzi, M., Wolf, S., Wullschlegel, J.: Pseudo-signatures, broadcast, and multi-party computation from correlated randomness. In: Franklin, M. (ed.) *CRYPTO 2004*. LNCS, vol. 3152, pp. 562–578. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-28628-8\\_34](https://doi.org/10.1007/978-3-540-28628-8_34)
20. Gohari, A.A., Anantharam, V.: Information-theoretic key agreement of multiple terminals: part I. *IEEE Trans. Inform. Theory* **56**(8), 3973–3996 (2010). <https://doi.org/10.1109/TIT.2010.2050832>
21. Hanaoka, G., Shikata, J., Zheng, Y., Imai, H.: Unconditionally secure digital signature schemes admitting transferability. In: Okamoto, T. (ed.) *ASIACRYPT 2000*. LNCS, vol. 1976, pp. 130–142. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-44448-3\\_11](https://doi.org/10.1007/3-540-44448-3_11)
22. Hanaoka, G., Shikata, J., Zheng, Y., Imai, H.: Efficient and unconditionally secure digital signatures and a security analysis of a multireceiver authentication code. In: Naccache, D., Paillier, P. (eds.) *PKC 2002*. LNCS, vol. 2274, pp. 64–79. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-45664-3\\_5](https://doi.org/10.1007/3-540-45664-3_5)
23. Khurana, D., Maji, H.K., Sahai, A.: Secure computation from elastic noisy channels. In: Fischlin, M., Coron, J.-S. (eds.) *EUROCRYPT 2016*. LNCS, vol. 9666, pp. 184–212. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49896-5\\_7](https://doi.org/10.1007/978-3-662-49896-5_7)
24. Lamport, L., Shostak, R., Pease, M.: The Byzantine general problem. *ACM Trans. Program. Lang. Syst.* **4**(3), 382–401 (1982). <https://doi.org/10.1145/357172.357176>

25. Maurer, U.: Secret key agreement by public discussion from common information. *IEEE Trans. Inform. Theory* **39**(3), 733–742 (1993). <https://doi.org/10.1109/18.256484>
26. Maurer, U., Wolf, S.: Secret-key agreement over unauthenticated public channels—part I: definitions and a completeness result. *IEEE Trans. Inform. Theory* **49**(4), 822–831 (2003). <https://doi.org/10.1109/TIT.2003.809563>
27. Maurer, U., Wolf, S.: Secret-key agreement over unauthenticated public channels—part II: privacy amplification. *IEEE Trans. Inform. Theory* **49**(4), 839–851 (2003). <https://doi.org/10.1109/TIT.2003.809559>
28. Maurer, U., Wolf, S.: Secret-key agreement over unauthenticated public channels—part II: the simulatability condition. *IEEE Trans. Inform. Theory* **49**(4), 832–838 (2003). <https://doi.org/10.1109/TIT.2003.809560>
29. Mitzenmacher, M., Upfal, E.: *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, Cambridge (2005). <https://doi.org/10.1017/CBO9780511813603>
30. Narayanan, V., Prabhakaran, V.M., Sangwan, N., Watanabe, S.: Complete characterization of broadcast and pseudo-signatures from correlations. *Cryptology ePrint Archive*, Paper 2023/233 (2023). <https://eprint.iacr.org/2023/233>
31. Nascimento, A.C.A., Winter, A.: On the oblivious-transfer capacity of noisy resources. *IEEE Trans. Inform. Theory* **54**(6), 2572–2581 (2008). <https://doi.org/10.1109/TIT.2008.921856>
32. Pfitzmann, B., Waidner, M.: Information-theoretic pseudosignatures and Byzantine agreement for  $t \geq n/3$ . IBM Research Technical Report RZ 2882 (#90830) (1996)
33. Prabhakaran, V., Prabhakaran, M.: Assisted common information with an application to secure two-party sampling. *IEEE Trans. Inform. Theory* **60**(6), 3413–3434 (2014). <https://doi.org/10.1109/TIT.2014.2316011>
34. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: 21st ACM STOC, pp. 73–85. ACM Press (May 1989). <https://doi.org/10.1145/73007.73014>
35. Renner, R., Wolf, S.: New bounds in secret-key agreement: the gap between formation and secrecy extraction. In: Biham, E. (ed.) *EUROCRYPT 2003*. LNCS, vol. 2656, pp. 562–577. Springer, Heidelberg (2003). <https://doi.org/10.1007/3-540-39200-9.35>
36. Renner, R., Wolf, S.: Unconditional authenticity and privacy from an arbitrarily weak secret. In: Boneh, D. (ed.) *CRYPTO 2003*. LNCS, vol. 2729, pp. 78–95. Springer, Heidelberg (2003). [https://doi.org/10.1007/978-3-540-45146-4\\_5](https://doi.org/10.1007/978-3-540-45146-4_5)
37. Tyagi, H., Watanabe, S.: Converses for secret key agreement and secure computing. *IEEE Trans. Inform. Theory* **61**, 4809–4827 (2015). <https://doi.org/10.1109/TIT.2015.2457926>
38. Tyagi, H., Watanabe, S.: *Information-Theoretic Cryptography*. Cambridge University Press, Cambridge (2023)
39. Wolf, S., Wullschlegel, J.: New monotones and lower bounds in unconditional two-party computation. In: Shoup, V. (ed.) *CRYPTO 2005*. LNCS, vol. 3621, pp. 467–477. Springer, Heidelberg (2005). [https://doi.org/10.1007/11535218\\_28](https://doi.org/10.1007/11535218_28)



# Privacy-Preserving Blueprints

Markulf Kohlweiss<sup>1</sup>(✉), Anna Lysyanskaya<sup>2</sup>, and An Nguyen<sup>2</sup>

<sup>1</sup> University of Edinburgh and Input Output, Edinburgh, UK  
markulf.kohlweiss@ed.ac.uk

<sup>2</sup> Brown University, Providence, USA  
{anna.lysyanskaya,an.q.nguyen}@brown.edu

**Abstract.** If everyone were to use anonymous credentials for all access control needs, it would be impossible to trace wrongdoers, by design. This would make legitimate controls, such as tracing illicit trade and terror suspects, impossible to carry out. Here, we propose a privacy-preserving blueprint capability that allows an auditor to publish an encoding  $\mathbf{pk}_A$  of the function  $f(x, \cdot)$  for a publicly known function  $f$  and a secret input  $x$ . For example,  $x$  may be a secret watchlist, and  $f(x, y)$  may return  $y$  if  $y \in x$ . On input her data  $y$  and the auditor's  $\mathbf{pk}_A$ , a user can compute an escrow  $Z$  such that anyone can verify that  $Z$  was computed correctly from the user's credential attributes, and moreover, the auditor can recover  $f(x, y)$  from  $Z$ . Our contributions are:

- We define secure  $f$ -blueprint systems; our definition is designed to provide a modular extension to anonymous credential systems.
- We show that secure  $f$ -blueprint systems can be constructed for all functions  $f$  from fully homomorphic encryption and NIZK proof systems. This result is of theoretical interest but is not efficient enough for practical use.
- We realize an optimal blueprint system under the DDH assumption in the random-oracle model for the watchlist function.

## 1 Introduction

Cryptography offers powerful answers on how to strike a balance between privacy and accountability. The study of anonymous credentials [2, 17–19, 27, 54, 55] has given us general practical tools that make it possible to obtain and prove possession of cryptographic credentials without revealing any additional information. In other words, users can obtain credentials without revealing who they are, and then prove possession of credentials in a way that is unlinkable to the session where these credentials were obtained and to other sessions in which they were shown. Anonymous credentials can be shown a limited number of times (compact e-cash) [15], or at a limited rate total or per verifier [14, 16]. Anonymous credentials are compatible with identity escrow [3, 51], where an appropriate trusted authority can establish the identity of the user when needed.

*In this paper, we extend the state-of-the-art on anonymous credentials by adding a new desirable feature: that of a privacy-preserving blueprint capability:*



*even a malicious authority cannot learn anything about a user other than what's revealed by comparing the blueprinted data with the user's data.*

Consider anonymous e-cash with a secret watchlist as a motivating application. In anonymous e-cash [15, 25, 26, 28], we have a Bank that issues e-coins (credentials), Users who withdraw and spend them, and Vendors (or Verifiers) that verify e-coins and accept them as payment in exchange for goods and services. Some small number of users are suspected of financial crimes, and, unbeknownst to them, a judge has placed them on a watchlist. We need a mechanism that allows an auditor to trace the transactions of these watchlisted users without violating the privacy of any other users, and also while keeping the contents of the watchlist confidential from everyone.

*A High-Level Definition of a Privacy-Preserving Blueprint.* We have three types of participants: the users, the verifiers, and the de-anonymization auditor. On input  $x$  (for example, a watchlist), the auditor outputs a blueprint  $\text{pk}_A$  that the users and verifiers will need.

Next, the user and the verifier engage in an anonymous transaction; we don't actually care what else happens in this transaction; the user might be proving to the verifier that they are authorized, or it may be an e-cash transaction. What we do care about is that, as a by-product of this transaction, the user and the verifier have agreed on a cryptographic commitment  $C$  such that (1) the user is in possession of the opening of  $C$ ; and (2) the transaction that just occurred guarantees that the opening of  $C$  contains user data  $y$  that is relevant for the auditor's needs. For example, imagine that  $x$  is a watchlist consisting of names of individuals of interest, and  $y$  contains a user's name; then this user is of interest to the auditor if  $y \in x$ .

To enhance this anonymous transaction with privacy-preserving blueprint capability, the user runs the algorithm **Escrow** to compute a value  $Z$  that is an escrow of the opening of the commitment  $C$ ; from  $Z$ , the auditor will be able to recover the information relevant to him, and no other information about the user. Specifically, in the watchlist scenario, the auditor will recover  $y$  if  $y \in x$ , but will learn nothing about the user if  $y \notin x$ . More generally, in an  $f$ -blueprint scheme, the auditor will recover  $f(x, y)$  and no additional information. The verifier's job is to verify the escrow  $Z$  against  $C$  using **VerEscrow** and only let the transaction go through if, indeed, it verifies.

It is important that even a malicious auditor cannot create a blueprint that corresponds to an unauthorized input  $x$ . To capture this, we also require that there is a publicly available cryptographic commitment  $C_A$ . Outside of our protocol, we expect a mechanism for arriving at an acceptable (but secret) input  $x$  and the commitment  $C_A$  to  $x$ . For example, a judge may publish a commitment to a secret watchlist, and privately reveal the opening to the auditor; or several authorities may be responsible for different components of a watchlist and the auditor aggregates them together in a publicly verifiable fashion; or another distributed protocol can be agreed upon for arriving at the commitment  $C_A$  such that its opening (i.e.,  $x$ ) is known to the auditor. To ensure that only such an authorized secret input  $x$  is blueprinted, a secure blueprint scheme must include

an algorithm  $\text{VerPK}$  that verifies that  $\text{pk}_A$  indeed corresponds to the value to which  $C_A$  is a commitment.

Our security definition mandates that the following properties hold: (1) correctness, so that honestly created blueprints and escrows pass  $\text{VerPK}$  and  $\text{VerEscrow}$ , respectively, and the escrow  $Z$  correctly decrypts; (2) soundness of  $\text{VerEscrow}$  that ensures that if, for a commitment  $C$ , escrow  $Z$  is accepted, then it correctly decrypts to  $f(x, y)$  where  $x$  is the opening of  $C_A$  and  $y$  is the opening of  $C$ ; (3) blueprint hiding, i.e., the blueprint  $\text{pk}_A$  does not reveal anything about  $x$  other than what the adversary can learn by forming valid escrows and submitting them for decryption; (4) privacy against a dishonest auditor that ensures that even if the auditor is malicious, an honest user's escrow contains no information beyond  $f(x, y)$ , where  $x$  is the opening of  $C_A$  and  $y$  is the opening of  $C$ ; and finally (5) privacy with an honest auditor that ensures that an adversary who does not control the auditor learns nothing from the escrows. We give a precise formal definition of an  $f$ -blueprint scheme in Sect. 3.

*Our Results.* Our first result is a blueprint scheme specifically for watchlists; more precisely, it is an  $f$ -blueprint scheme for

$$f(x, y) = \begin{cases} y & \text{if } y = y_1 \| y_2 \text{ and } y_2 \in x \\ \perp & \text{otherwise} \end{cases}$$

where  $y_1$  denotes  $O(\log \lambda)$  most significant bits of  $y$ . This first scheme is secure in the random-oracle model under the decisional Diffie-Hellman assumption. The size of  $\text{pk}_A$  is optimal at  $O(\lambda n)$  where  $\lambda$  is the security parameter, which is linear in the number of bits needed to represent a group element; and the watchlist  $x$  consists of  $n$  elements of  $\mathbb{Z}_q$ , where  $q$  ( $\log q = \Theta(\lambda)$ ) is the order of the group. The size of the escrow  $Z$  is also  $O(\lambda n)$ .

Our second result is an  $f$ -blueprint scheme for any  $f$  from fully homomorphic encryption (FHE) and non-interactive zero-knowledge proofs of knowledge. In the full version of this paper [52], we also show how to obtain an  $f$ -blueprint scheme for any  $f$  from non-interactive secure computation (NISC) [50].

*Technical Roadmap.* We obtain the results above via the same general method: by first defining (Sect. 4) and then realizing (Sects. 6 and 7) a homomorphic-enough cryptosystem (HEC) for the function  $f$ . We can think of a homomorphic-enough cryptosystem as a protocol between Alice and Bob that works as follows: first, Alice uses the  $\text{HECENC}$  algorithm to encode her input  $x$  into a value  $X$ , and she also obtains a decryption key  $d$  for future use; next, Bob uses  $\text{HECEVAL}$  to compute an encryption  $Z$  of  $z = f(x, y)$  from Alice's encoding  $X$  and his input  $y$ . Finally, Alice runs  $\text{HECDEC}$  to recover  $z$  from  $Z$ . To be useful for our application, an HEC scheme must be correct even when the inputs to the algorithms are chosen maliciously, and it also must ensure that  $X$  hides  $x$ , and that  $X$  and  $Z$  together hide the inputs  $x$  and  $y$ . Additionally, it must allow for an algorithm  $\text{HECDIRECT}$  that computes an encryption  $Z$  of  $z$  directly from  $X$  and  $z = f(x, y)$ , such that its output is indistinguishable from the output of  $\text{HECEVAL}$ , even if Alice is malicious.

A HEC combined with an appropriate non-interactive zero-knowledge (NIZK) proof system gives a generic construction of an  $f$ -blueprint. The auditor obtains  $(X, d) \leftarrow \text{HECENC}(x)$ , and an NIZK proof  $\pi_A$  that  $X$  was computed correctly in a way that corresponds to the opening of  $C_A$ ; he sets the blueprint as  $\text{pk}_A = (X, \pi_A)$ . Verifying this blueprint amounts to verifying  $\pi_A$ . To compute the escrow  $Z$ , the user obtains  $Z' \leftarrow \text{HECEVAL}(X, y)$  and then a proof  $\pi_Z$  that  $Z'$  was computed correctly from  $X$  and the opening of  $C$ ; then set  $Z = (Z', \pi_Z)$ . Verifying the escrow amounts to verifying  $\pi_Z$ . Finally, in order to recover  $f(x, y)$  from the escrow  $Z$ , the auditor uses the decryption key  $d$  to run  $\text{HECDEC}(d, Z')$ .

Given this roadmap, our theoretical construction that works for any  $f$  is relatively straightforward: we show that HEC can be realized from circuit-private fully homomorphic encryption [35, 46, 60] which, in turn, can be realized from regular fully homomorphic encryption [9, 10, 46, 47]. The circuit-privacy guarantee ensures that  $Z$  hides Bob’s input  $y$  from a malicious Alice. Alternatively, as we explore in the full version of this paper [52], it can be realized for any  $f$  from a related primitive of non-interactive secure computation (NISC) [50]. Since here we don’t aim for efficiency, general (inefficient) simulation-extractable NIZK PoK can be used for the proofs. This instantiation of our generic construction is presented in Sect. 7.

Our practical construction for watchlists under the decisional Diffie-Hellman assumption is not as straightforward: first, it requires that we construct a practical homomorphic enough cryptosystem based on DDH, and next we need efficient non-interactive zero-knowledge proof systems for computing and verifying  $\pi_A$  and  $\pi_Z$ . Let us give a brief overview.

Our HEC construction uses the ElGamal cryptosystem [36, 65] as a building block. Suppose as part of setup we are given a group  $\mathbb{G}$  of order  $q$  in which the decisional Diffie-Hellman assumption holds. Let  $g$  be a generator of  $\mathbb{G}$ . In order to encode her input  $x = (a_1, \dots, a_n)$ , Alice’s HECENC algorithm first generates an ElGamal key pair  $(\text{pk}, \text{sk})$ . She then picks a random  $s \leftarrow \mathbb{Z}_q^*$  and computes the coefficients  $c_0, \dots, c_n$  of the  $n$ -degree polynomial  $p(\mathbf{x}) = s \prod (\mathbf{x} - a_i)$  for which  $a_1, \dots, a_n$  are the  $n$  zeroes. The encoding  $X$  are ElGamal encryptions  $C_0, \dots, C_n$  of the values  $g^{c_0}, \dots, g^{c_n}$  under the ElGamal public key  $\text{pk}$ , so the output of HECENC is  $X = (C_0, \dots, C_n, \text{pk})$ , and  $d = \text{sk}$ .

Bob’s algorithm HECEVAL computes  $Z$  as follows: first, it parses  $y = y_1 || y_2$  (recall that  $y_1$  denotes the first  $O(\log \lambda)$  bits of  $y$ ). Then Bob obtains an ElGamal encryption  $E$  of  $g^{p(y_2)}$  from the encrypted coefficients  $C_0, \dots, C_n$ : since the ElGamal cryptosystem is multiplicatively homomorphic,  $E = C_0 C_1^{y_2} C_2^{y_2^2} \dots C_n^{y_2^n}$  is the desired ciphertext (for an appropriate multiplication operation on ElGamal ciphertexts). Next, let  $F$  be an encryption of  $g^y$ ; finally, Bob obtains the ciphertext  $Z = FE^r$ , i.e., Bob uses  $E$  to mask the encryption of  $g^y$ ; if  $E$  is an encryption of 0, the mask won’t work and  $Z$  will decrypt to  $g^y$ .

This is reminiscent of the private set intersection construction of Freedman, Nissim and Pinkas [42], but with a subtle difference: the polynomial encoded as part of  $X$  has an additional random coefficient,  $s$ . Thus, even if Bob knows Alice’s entire input  $x$ , he still does not know  $p(a)$  for  $a \notin x$ . This ensures that in

the event that  $f(x, y) = \perp$ , Bob cannot set  $r$  in such a way that  $Z$  will decrypt to a value of his choice; instead, it will decrypt to a random value.

Finally,  $\text{HEC}_{\text{DEC}}(d, Z)$  decrypts the ElGamal ciphertext  $Z$  to some group element  $u \in \mathbb{G}$ , and for each  $a_i \in x$ , and for all possible values for  $y_1$ , checks whether  $g^{y_1 \parallel a_i} = u$ . If it finds such a pair, it outputs it; else, it outputs  $\perp$ .

Plugging in this HEC scheme in our generic construction gives us an efficient blueprint scheme for watchlists as long as we can also find efficient instantiations of the NIZK proof systems for computing the proofs  $\pi_A$  and  $\pi_Z$ . As was already well-known [23, 44, 58], we can represent the statement that a given ElGamal ciphertext encrypts  $g^a$  such that a given Pedersen [62] commitment  $C$  is a commitment to  $a$  as a statement about equality of discrete logarithm representations; moreover, we can also represent statements about polynomial relationships between committed values (i.e., that  $C_p$  is a commitment to the value  $p(a_1, \dots, a_\ell)$  where  $p$  is a polynomial, and commitments  $C_1, \dots, C_\ell$  are to values  $a_1, \dots, a_\ell$ ) as statements about equality of representations. Using this fact, as well as the fact that efficient NIZK proofs of knowledge for equality of discrete logarithm representations in the random-oracle model are known [31, 43, 44], we can also efficiently instantiate the NIZK proof system in the random-oracle model.

A subtlety in using these random-oracle-based proof systems, however, is that generally such proof systems' knowledge extractors require black-box access to the adversary and involve rewinding it. In situations where the adversary expects to also issue queries to its challenger, and a security experiment or reduction must extract the adversary's witness in order to answer them, using such proof systems runs into the nested rewinding problem. One could opt to use straight-line extractable proofs instead (in such proofs, the knowledge extractor does not need to rewind the adversary); however, known techniques to achieve straight-line extraction come either at a  $\omega(\log \lambda)$  multiplicative cost [13, 39] or require cumbersome setup assumptions [21].

A more efficient technique is to have a common random string (CRS) and interpret it as an ElGamal public key. There is an efficient  $\Sigma$ -protocol [31] for proving that the contents of two ElGamal ciphertexts under two different keys are equal; it can be converted into a non-interactive zero-knowledge proof using the Fiat-Shamir heuristic [38] in the random-oracle model. If one of these public keys comes from the CRS, then the soundness of the proof system allows for straight-line extraction that uses the corresponding secret key as the extraction trapdoor. Here we give a formalization of this previously used (e.g. [20]) approach.

Specifically, we formulate a new flavor of NIZK proof of knowledge systems: black-box extractability with partial straight-line (BB-PSL) extraction, and give an efficient NIZK BB-PLS PoK proof system for equality of discrete-logarithm representations. This proof system allows straight-line extraction (i.e. extraction from the proof itself, without rewinding the adversary) of a function of the witness (for example, instead of extracting  $w$  the extractor computes  $g^w$ ); this gives the security experiment enough information to proceed. Although this approach is somewhat folklore, we believe our rigorous formulation and instantiation in the random-oracle model (Sect. 2.2) may be of independent interest.

*How Our Scheme Builds on the Anonymous Credentials Literature.* Note that, as stated so far, neither the definitions nor the schemes concern themselves with credentials. Instead, the user and the verifier agree on a commitment  $C$  to the user's relevant attribute  $y$ . Out of band, the user may have already convinced the verifier that she has a credential from some third-party organization attesting that  $y$  is meaningful. For example, if  $y$  is the user's name, then the third-party organization might be the passport bureau. Indeed, this is how anonymous credentials work in general [2, 18, 19, 54], and therefore this modeling of the problem allows us to add this feature to anonymous credentials in a modular way. Moreover, our ElGamal-based scheme is compatible with literature on anonymous credentials [2, 18, 19, 54] and compact e-cash and variants [14–16] because Pedersen commitments are used everywhere.

*Related Work.* Group signatures and identity escrow schemes [1, 5, 22, 29, 51] allow users to issue signatures anonymously on behalf of a group such that an anonymity-revoking trustee can discover the identity of the signer. The difference between this scenario and what we are doing here is that in group signatures the signer's identity is always recoverable by the trustee, while here it is only recoverable if it matches the watchlist.

Group signatures with message-dependent opening [63] and bifurcated [53], multimodal and related signatures [34, 41, 59] allow a tracing authority to recover a function of the user's private information that's known to the user at the time of group-signing or credential showing. In contrast, in a secure blueprinting scheme, the user knows only one of two inputs to this function.

Another related line of work specifically for watchlists is private set intersection (PSI) [24, 42]. Although techniques from PSI are helpful here, in general PSI is an interactive two-party protocol, while here, the Auditor who knows the watchlist  $x$  is offline at the time when the user is forming the escrow. Private searching on streaming data [61] allows an untrusted proxy to process streaming data using encrypted keywords. The resulting encrypted data does not come with any assurance that it was *correct*. In contrast, in our scenario, the verifier and the auditor can both verify that  $Z$  was computed correctly.

A series of recent papers explored accountable law enforcement access system [40, 48, 49, 64]. None of them, however, consider integration with anonymous credential systems for privacy-preserving authentication. Break-glass encryption by Scafuro [64] realizes a mechanism in which the auditor can decrypt Alice's ciphertexts simply by reliably revealing that he did so, i.e., that he broke the glass. The choice of which messages are to be leaked can happen even after Alice's public key is generated. Scafuro achieves this in certain strong models, such as those of hardware tokens and existence of a blockchain. In abuse-resistant law enforcement access systems (ARLEAS) [49], a law enforcement agency with a valid warrant can secretly place a user Alice under surveillance. They will be able to decrypt messages that are encrypted to Alice's public key, but not those encrypted to other users for whom surveillance has not been authorized. Moreover, ARLEAS make it possible for an email server to enforce compliance by verifying that an encrypted message indeed allows lawful access by law enforcement;

and (in a nutshell) all participants can verify the validity of all warrants even though they are unable to tell who is under surveillance. In our view, ARLEAS follows principles that are similar to ours: finding a way to reconcile the need to monitor illegal activity with privacy needs of the law-abiding public. However, since ARLEAS concerns itself with encryption, while we worry about privacy-preserving authentication, our technical contributions are somewhat orthogonal.

## 2 Preliminaries

*The ElGamal Cryptosystem and Its Security.* Let KGen be an algorithm that, on input a description of a group  $\mathbb{G}$  with generator  $g$  of prime order  $q$  in which the discrete-logarithm problem is hard for PPT (in the security parameter  $1^\lambda$ ) adversaries, outputs (1) a public key  $\mathbf{pk}$  consisting of an element  $y \leftarrow_{\$} \mathbb{G}$ ; and (2) a secret key  $\mathbf{sk} = s$  such that  $g^s = y$ . The encryption algorithm Enc encrypts a message  $m \in \mathbb{G}$  by sampling  $r \leftarrow_{\$} \mathbb{Z}_q$  and outputting the ciphertext  $c = (g^r, my^r)$ . The decryption algorithm Dec decrypts  $c = (a, b) \in \mathbb{G}^2$  by computing  $ba^{-s}$ . We use  $c \stackrel{r}{\leftarrow} \text{Enc}(\mathbf{pk}, m)$  and  $\text{Enc}(\mathbf{pk}, m; r)$  to make randomness explicit.

ElGamal is semantically secure under the decisional Diffie-Hellman assumption [66]. In this paper, we use the equivalent notion of security against chosen plaintext attack (IND-CPA) formulated by Boneh and Shoup [7]. In their security game, the adversary continuously interacts with either the 0-encryption oracle that always encrypts the first of the two messages the adversary sends it, or with the 1-encryption oracle that always encrypts the second message. Their security definition is more convenient for us because it allows us to avoid an additional hybrid argument.

Let  $\oplus : \mathbb{G}^2 \times \mathbb{G}^2 \rightarrow \mathbb{G}^2$  be the operator for the homomorphic composition of two ElGamal ciphertexts  $c_1 = (a_1, b_1) \in \mathbb{G}^2, c_2 = (a_2, b_2) \in \mathbb{G}^2$  such that:  $c_1 \oplus c_2 := (a_1 \cdot a_2, b_1 \cdot b_2)$  where  $\cdot$  is the group operator of  $\mathbb{G}$ . We also write  $c^a$  as shorthand for repeated operation of  $c$  with itself  $a$  times.

**Definition 1 (Statistically hiding non-interactive commitment).** *A pair of algorithms (CSetup, Commit) constitute a statistically hiding non-interactive commitment scheme for message space  $M_{cpar}$  and randomness space  $R_{cpar}$  if they satisfy (1) statistical hiding, i.e., for any  $cpar$  output by CSetup( $1^\lambda$ ), for any  $m_0, m_1 \in M_{cpar}$ , the distributions  $D(cpar, m_0)$  and  $D(cpar, m_1)$  are statistically close, where  $D(cpar, m) = \{r \leftarrow R_{cpar} : \text{Commit}_{cpar}(m; r)\}$ ; and (2) computational binding, i.e. for any PPT adversary  $\mathcal{A}$ , there exists a negligible  $\nu$  such that  $\Pr[cpar \leftarrow \text{CSetup}(1^\lambda); (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(cpar) : \text{Commit}_{cpar}(m_0; r_0) = \text{Commit}_{cpar}(m_1; r_1) \wedge m_0 \neq m_1] = \nu(\lambda)$*

We will use the Pedersen commitment scheme which employs a cyclic group  $\mathbb{G}$  of prime order  $q$ . Let  $g, h_1, h_2, \dots, h_n$  be generators of  $\mathbb{G}$  and  $m_1, m_2, \dots, m_n \in \mathbb{Z}_q^n$ , then  $\text{Commit}_{h_1, h_2, \dots, h_n, g}(m_1, m_2, \dots, m_n)$  samples  $r \leftarrow_{\$} \mathbb{Z}_q$  and computes  $g^r \prod_{i=1}^n h_i^{m_i}$ . This scheme is binding under the discrete logarithm assumption in  $\mathbb{G}$ . We write  $\text{Commit}_{h_1, \dots, h_n, g}(m_1, \dots, m_n; r)$  to make randomness explicit.

### 2.1 Non-interactive Zero Knowledge

Let  $\mathcal{R}$  be a polynomial-time verifiable binary relation. For a pair  $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ , we refer to  $\mathbf{x}$  as the statement and  $\mathbf{w}$  as the witness. Let  $\mathcal{L} = \{\mathbf{x} \mid \exists \mathbf{w} : (\mathbf{x}, \mathbf{w}) \in \mathcal{R}\}$ .

A non-interactive proof system for  $\mathcal{R}$  consists of a prover algorithm  $P$  and verifier algorithm  $V$  both given access to a setup  $S$ . The setup can either be a random oracle or a reference string—we show later how we abstract over the differences in their interfaces.  $P$  takes as input a statement  $\mathbf{x}$  and witness  $\mathbf{w}$ , and outputs a proof  $\pi$  if  $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$  and  $\perp$  otherwise.  $V$  takes as input  $(\mathbf{x}, \pi)$  and either outputs 1 or 0.

**Definition 2 (NIZK).** *Let  $S$  be the setup, and  $(P, V)$  be a pair of algorithms with access to setup  $S$ .  $\Phi = (S, P, V)$  is a simulation-sound (optionally extractable) non-interactive zero-knowledge proof system for relation  $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$  if it has the following properties:*

**Completeness:** For all  $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ ,  $\Pr[\pi \leftarrow P^S(\mathbf{x}, \mathbf{w}) : V^S(\mathbf{x}, \pi) = 0] = 0$ .

$S$  is a stateful oracle that captures both the common-random-string setting and the random-oracle setting. In the random-oracle setting,  $S$  responds to a query  $m$  by sampling a random string  $h$  of appropriate length  $\ell$  (clear from context). In the common-reference-string (CRS) setup, it samples a reference string on the first invocation, and from then onward returns the same reference string to all callers.

**Zero-Knowledge:** The zero-knowledge property requires that no adversary can distinguish the real game in which the setup is generated honestly and an honest prover computes proofs using the correct algorithm  $P$ , from the simulated game in which the proofs are computed by a simulator that does not take witnesses as inputs, and in which the setup is also generated by the simulator. More formally, there exist probabilistic polynomial time (PPT) simulator algorithms  $(\text{SimS}, \text{Sim})$  such that, for any PPT adversary  $\mathcal{A}$  interacting in the experiment in Fig. 1, the advantage function  $\nu(\lambda)$  defined below is negligible:

$$\text{Adv}_{\mathcal{A}}^{\text{nizk}}(\lambda) = \left| \Pr\left[\text{NIZK}^{\mathcal{A},0}(1^\lambda) = 0\right] - \Pr\left[\text{NIZK}^{\mathcal{A},1}(1^\lambda) = 0\right] \right| = \nu(\lambda)$$

$\text{NIZK}^{\mathcal{A},0}(1^\lambda)$	$O_S(m)$	$O_P(\mathbf{x}, \mathbf{w})$
$\text{return } \mathcal{A}^{S(\cdot), P^S(\cdot, \cdot)}(1^\lambda)$	$\text{state}, h, \tau_{\text{Ext}} \leftarrow \text{SimS}(\text{state}, m)$	$\text{if } (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} : \text{return } \perp$
$\text{NIZK}^{\mathcal{A},1}(1^\lambda)$	$\text{return } h$	$\text{state}, \pi \leftarrow \text{Sim}(\text{state}, \mathbf{x})$
$\text{return } \mathcal{A}^{O_S, O_P(\cdot, \cdot)}(1^\lambda)$		$\text{return } \pi$

Fig. 1. NIZK game

SimS shares state with Sim modeling both RO programming and CRS trapdoors. Additionally there is an extraction trapdoor  $\tau_{\text{Ext}}$  that will be used below to define simulation extractability.

**Soundness:** A proof system is sound if no adversary can fool a verifier into accepting a proof of a false statement. It is simulation sound if the adversary cannot do so even given oracle access to the simulator—of course in that case the adversary is prohibited from outputting statement-proof pairs for which the proof was obtained from the simulator. It is a proof of knowledge if a knowledge extractor algorithm can compute the witness given appropriate access to the adversarial prover’s algorithm. We explore various flavors of simulation soundness in the full version of this paper [52], but here we focus on just one of them: the flavor of a proof of knowledge that allows for (partial) straight-line extraction.

## 2.2 NIZK Proof of Knowledge

**Simulation Extractability:** A proof system is extractable (also often called a *proof of knowledge*, or PoK for short) if there exists a polynomial-time extractor algorithm that, on input a proof  $\pi$  for a statement  $\mathfrak{x}$  that passes verification, outputs the witness  $w$  for  $\mathfrak{x}$ . In order to reconcile extractability with the zero-knowledge property, it is important that the extractor algorithm Ext have some additional information that is not available to any regular participants in the system. This information depends on the setup  $S$ : in the CRS setting, it is a trapdoor that corresponds to the CRS; in the random-oracle setting it comes from the ability to observe the adversary’s queries to the random oracle. Note that, in addition, trapdoors can be embedded by programming the random oracle. Further, a proof system is simulation-extractable if the extractor algorithm works even when the adversary has oracle access to the simulator and can thus obtain simulated proofs.

Let  $Q$  denote the simulator’s query tape that records all the queries the adversary  $\mathcal{A}$  made to the simulator.  $Q_S$  denotes the setup query tape that records the queries, replies, and embedded trapdoors of the simulated setup; this is explicitly recorded by  $O_S$  and  $\tilde{O}_S$ . As we will discuss below,  $\tilde{O}_S$  additionally reveals to  $\mathcal{A}$  the extraction trapdoor  $\tau_{\text{Ext}}$ ; this captures adaptive extraction from many proofs.

An attractive definition of simulation extractability is the one of straight-line extractability [39]: the extractor obtains the witness just from  $Q_S$  and the pair  $(\mathfrak{x}, \pi)$ . A weaker definition allows for black-box extractability, where the extractor additionally obtains black-box access to  $\mathcal{A}$ , i.e. it can reset it to a previous state. By  $\text{BB}(\mathcal{A})$  we denote this mode of access to  $\mathcal{A}$ , and by  $\text{Ext}^{\text{BB}(\mathcal{A})}(Q_S, \mathfrak{x}, \pi)$  we denote an extractor algorithm that, in addition to its inputs, also has this access to  $\mathcal{A}$ . See the full version of this paper for additional discussions and the definition of the black-box simulation extractability game  $\text{NISimBBExtract}$ . We now propose a notion that falls between straight-line and black-box simulation extractability.



**Black-Box with Partial Straight Line (BB-PSL) Simulation**

**Extractability:** Sometimes, it is good enough that a straight-line extractor be able to learn something about the witness, say some function  $f(\mathbb{w})$ , not necessarily the entire witness. For such a scenario, it is convenient to have two extractors:  $\text{Ext}$  that is a black-box extractor that extracts the entire witness given black-box access to the adversary, and  $\text{ExtSL}$  that extracts some function of that witness in a straight-line fashion. The reason this is good enough for some proofs of security is that, in a reduction,  $f(\mathbb{w})$  may be enough information for the reduction to know how to proceed, without the need to reset the entire security experiment. This is similar to  $f$ -extractability [4].

Let us now formalize BB-PSL simulation extractability; let  $\Phi = (\mathbb{S}, \mathbb{P}, \mathbb{V})$  be an NIZK proof system satisfying the zero-knowledge property above; let  $(\text{SimS}, \text{Sim})$  be the simulator. Let  $f$  be any polynomial-time computable function.  $\Phi$  is  $f$ -BB-PSL simulation-extractable if there exists a pair of polynomial-time extractor algorithms  $(\text{Ext}, \text{ExtSL})$  such that for any PPT adversary  $\mathcal{A}$  participating in the game defined in Fig. 2, the advantage function  $\nu(\lambda)$  defined below is negligible. As mentioned before,  $\mathcal{Q}$  denotes the query tape.  $\mathcal{Q}_{\text{Ext}}$  denotes the setup query tape that records the queries, replies, and embedded trapdoors of the simulated setup; this is explicitly recorded by  $\mathcal{O}_{\mathbb{S}}$ .

$$\text{Adv}_{\mathcal{A}}^{\text{nisimbbpslextract}}(\lambda) = \Pr \left[ f\text{-NISimBBPSLExtract}^{\mathcal{A}}(1^\lambda) = 1 \right] = \nu(\lambda)$$

for some negligible function  $\nu$ .

$f\text{-NISimBBPSLExtract}^{\mathcal{A}}(1^\lambda)$	
1 : $\mathcal{Q}, \mathcal{Q}_{\mathbb{S}} \leftarrow [ ]$	
2 : $(\mathbb{x}, \pi) \leftarrow \mathcal{A}^{\tilde{\mathcal{O}}_{\mathbb{S}}(\cdot), \mathcal{O}_{\text{Sim}}(\cdot)}(1^\lambda)$	
3 : $\mathbb{w} \leftarrow \text{Ext}^{\text{BB}(\mathcal{A})}(\mathcal{Q}_{\mathbb{S}}, \mathbb{x}, \pi)$	
4 : $\mathbb{w}' \leftarrow \text{ExtSL}(\mathcal{Q}_{\mathbb{S}}, \mathbb{x}, \pi)$	
5 : <b>return</b> $\mathbb{V}^{\mathcal{O}_{\mathbb{S}}}(\mathbb{x}, \pi) \wedge (\mathbb{x}, \pi) \notin \mathcal{Q} \wedge (\mathbb{x}, \mathbb{w}) \notin \mathcal{R} \wedge \mathbb{w}' \neq f(\mathbb{w})$	
$\mathcal{O}_{\mathbb{S}}(m) \quad \tilde{\mathcal{O}}_{\mathbb{S}}(m)$	$\mathcal{O}_{\text{Sim}}(\mathbb{x})$
1 : $\text{state}, h, \tau_{\text{Ext}} \leftarrow \text{SimS}(\text{state}, m)$	1 : $\text{state}, \pi \leftarrow \text{Sim}(\text{state}, \mathbb{x})$
2 : $\mathcal{Q}_{\mathbb{S}}.\text{add}((m, h, \tau_{\text{Ext}}))$	2 : $\mathcal{Q}.\text{add}((\mathbb{x}, \pi))$
3 : <b>return</b> $h, \tau_{\text{Ext}}$	3 : <b>return</b> $\pi$

**Fig. 2.**  $f$ -NISimBBPSLExtract game

**More on the Simulator and Extractor.** In the games NIZK, NISimSound, and NISimBBPSLExtract the simulator initializes and updates the setup using SimS and then responds to queries from  $\mathcal{A}$  for simulated proofs using Sim. Note that the two halves of the simulator, SimS and Sim, share state information,

and update it when queried. This captures both the CRS and the random-oracle settings. In the CRS setting,  $\text{SimS}$  computes the reference string  $S$  so that it can pass the corresponding simulation trapdoor to  $\text{Sim}$  via the shared `state`. In the random-oracle (RO) setting,  $\text{SimS}$  programs the random oracle (computes the value  $h$  that the random oracle will return when queried on  $m$ ) and uses the shared `state` in order to memorize the information that  $\text{Sim}$  will need to use  $h$  in the future. Similarly, in the random-oracle mode,  $\text{Sim}$  has the ability to program the random oracle as well and memorize what it did using the `state` variable.

In the simulation extractability experiments, the extractor  $\text{Ext}$  takes  $\mathcal{Q}_S$  as input. In the CRS model,  $\mathcal{Q}_S$  will contain the extraction trapdoor corresponding to the CRS. In the RO model,  $\mathcal{Q}_S$  also contains information that the simulator algorithms  $\text{SimS}$  generated, such as how the RO was programmed and where the adversary queried it. It does not, however, contain the simulation trapdoor or give the extractor the ability to program the RO.

Our definition requires successful extraction even when all information in  $\mathcal{Q}_S$ , in particular  $\tau_{\text{Ext}}$ , is available to the adversary. This allows the adversary to run  $\text{Ext}$  itself, and thus allows for extraction from multiple proofs.

**Instantiating Simulation Extractable Proofs.** While simulation-extractable proof systems exist for all NP relations [33], there are multiple ways to realize non-interactive zero-knowledge (NIZK) proof systems more efficiently. One of them is to start with  $\Sigma$ -protocols and convert them into a NIZK proof in the random oracle model, e.g. using the techniques of [37,38,56]. As we will elaborate below,  $\Sigma$  protocols are particularly suitable for proving knowledge of group isomorphisms such as discrete logarithm representations; see, e.g. [57]. They can also efficiently prove disjunctive statements [30]. This has been used for range proofs.

Bulletproofs [12] is a practically efficient NIZK proof system for arithmetic circuits, specifically optimized for range-proofs. Recent work shows that Bulletproofs are simulation extractable [45] and can be integrated with  $\Sigma$ -protocols [11].

Bernhard et al. [6, Theorem 1] state that Fiat-Shamir  $\Sigma$ -protocols are black-box simulation extractable with respect to expected polynomial-time adversaries. To show partial straightline extractability we use a theorem of [37, Theorem 2] that shows that  $\Sigma$ -protocols compiled using Fiat-Shamir are simulation-sound and adapt the theorem of [32, Theorem F.1] which shows how to transform simulation-sound into simulation-extractable NIZK, by encrypting the witness to the sky. Our approach differs from their approach in that we only encrypt a partial witness and can thus use groups for which computing discrete logarithms is hard.

In Sect. 2.5 we give a construction from  $\Sigma$ -protocols of a proof system  $\Psi$  for equality of discrete logarithm representation relations and prove that it is an  $f$ -BB-PSL simulation-extractable NIZK proof system in the random-oracle model for an appropriate  $f$ .

*Notation.* When using NIZK proofs of knowledge in a protocol, it is convenient to be able to compactly specify what exactly the prover is proving its knowledge of. We shall use the notation:

$$\pi \leftarrow \text{PoK}_{\Psi} \left\{ \mathbf{w} : R(\mathbf{x}, \mathbf{w}) \right\}$$

to indicate that the proof  $\pi$  was computed as follows: the proof system  $\Psi = (\mathbf{S}, \mathbf{P}, \mathbf{V})$  for the relation  $R$  was used; the prover ran  $\mathbf{P}^{\mathbf{S}}(\mathbf{x}, \mathbf{w})$ ; to verify  $\pi$ , the algorithm  $\mathbf{V}^{\mathbf{S}}(\mathbf{x}, \pi)$  needs to be run. In other words, the value  $\mathbf{w}$  in this notation is the witness the knowledge of which the prover is proving to the verifier, while  $\mathbf{x}$  is known to the verifier. A helpful feature of this notation is that it describes what we need  $\Psi$  to be: it needs to be a NIZK PoK for the relation  $R$ .

### 2.3 $\Sigma$ -Protocol for Proof of Equality of Discrete Logarithm Representations

Let  $R_{\text{eqrep}}$  be the following relation:  $R_{\text{eqrep}}(\mathbf{x}, \mathbf{w})$  accepts if  $\mathbf{x} = (\mathbb{G}, \{x_i, \{g_{i,1}, \dots, g_{i,m}\}_{i=1}^n\})$  where  $\mathbb{G}$  is the description of a group of order  $q$ , and all the  $x_i$ s and  $g_{i,j}$ s are elements of  $\mathbb{G}$ , and witness  $\mathbf{w} = \{w_j\}_{j=1}^m$  such that  $x_i = \prod_{j=1}^m g_{i,j}^{w_j}$ .

**P**→**V**. On input the  $(\mathbf{x}, \mathbf{w}) \in R_{\text{eqrep}}$ , the Prover chooses  $e_j \leftarrow \mathbb{Z}_q$  for  $1 \leq j \leq m$  and computes  $d_i = \prod_{j=1}^m g_{i,j}^{e_j}$  for  $1 \leq i \leq n$ . Finally, the Prover sends to the Verifier the values  $\text{com} = (d_1, \dots, d_n)$ .

**P**←**V**. On input  $\mathbf{x}$  and  $\text{com}$ , the Verifier responds with a challenge  $\text{chal} = c$  for  $c \leftarrow \mathbb{Z}_q$ .

**P**→**V**. The Prover receives  $\text{chal} = c$  and computes  $s_i = e_i + cw_i \bmod q$  for  $1 \leq i \leq m$ , and sends  $\text{res} = (s_1, \dots, s_m)$  to the Verifier.

**Verification.** The Verifier accepts if for all  $1 \leq i \leq n$ ,  $d_i x_i^c = \prod_{j=1}^m g_{i,j}^{s_j}$ ; rejects otherwise.

**Simulation.** On input  $\mathbf{x}$  and  $\text{chal} = c$ , the simulator chooses  $s_j \leftarrow \mathbb{Z}_q$  for  $1 \leq j \leq m$ , and sets  $d_i = (\prod_{j=1}^m g_{i,j}^{s_j}) / x_i^c$  for  $1 \leq i \leq n$ . He then sets  $\text{com} = (d_1, \dots, d_n)$  and  $\text{res} = (s_1, \dots, s_m)$ .

**Extraction.** On input two accepting transcripts for the same  $\text{com} = (d_1, \dots, d_n)$ , namely  $\text{chal} = c$ ,  $\text{res} = (s_1, \dots, s_m)$ , and  $\text{chal}' = c'$ ,  $\text{res}' = (s'_1, \dots, s'_m)$ , output  $w_j = (s_j - s'_j) / (c - c') \bmod q$  for  $1 \leq j \leq m$ .

### 2.4 From $\Sigma$ -Protocols to BB Simulation Extractable NIZK PoK via Fiat-Shamir

Let  $\Psi_{\text{eqrep}} = (\mathbf{S}_{\text{eqrep}}, \mathbf{P}_{\text{eqrep}}, \mathbf{V}_{\text{eqrep}})$  be the proof system we get from the  $\Sigma$ -protocol described in Sect. 2.3 via the Fiat-Shamir heuristic. Specifically,  $\mathbf{S}_{\text{eqrep}}$  is a random oracle.

We use a theorem of [37, Theorem 2] that shows that  $\Sigma$ -protocols compiled using Fiat-Shamir are simulation-sound; moreover, it follows from a theorem of [6, Theorem 1] and the proof of [37, Theorem 3] that it is in fact black-box simulation extractable.

Recall that the notation  $\pi \leftarrow \text{PoK}_{\Psi_{\text{eqrep}}} \left\{ \mathbf{w} : R_{\text{eqrep}}(\mathbf{x}, \mathbf{w}) \right\}$  denotes that the proof  $\pi$  is the output of  $\mathbf{P}_{\text{eqrep}}$ .

**2.5  $g^x$ -BB-PSL Simulation Extractable NIZK from  $\Psi_{\text{eqrep}}$**

Now we want a BB-PSL simulation extractable proof system for  $R_{\text{eqrep}}$  such that, in a straight-line fashion, a function of  $w$  can be extracted. Specifically, recall that  $x = (\mathbb{G}, \{x_i, \{g_{i,1}, \dots, g_{i,m}\}\}_{i=1}^n)$  and  $w = \{w_j\}_{j=1}^m$  such that  $x_i = \prod_{j=1}^m g_{i,j}^{w_j}$ .

Consider the following proof system  $\Psi = (\text{S}, \text{P}, \text{V})$  for the relation  $R_{\text{eqrep}}$  and for the function  $f(J, \cdot)$ , defined as follows. Let  $g$  be the generator of  $\mathbb{G}$  included in the description of  $\mathbb{G}$ . Let  $J$  be a subset of the set of indices  $[m]$ . Let  $f(J, w) = \{g^{w_j} : j \in J\}$ .

$\text{S}$  is a random oracle, but we interpret its output as follows: On input the description of a group  $\mathbb{G}$  with generator  $g$  of order  $q$ , outputs a random element  $h$  of  $\mathbb{G}$ ; we can think of this  $h$  as the public key of the ElGamal cryptosystem.

$\text{P}$  works as follows: on input  $x = (\mathbb{G}, \{x_i, \{g_{i,1}, \dots, g_{i,m}\}\}_{i=1}^n)$  and  $w = \{w_j\}_{j=1}^m$ , it first obtains  $h = \text{S}(\mathbb{G})$  and then forms the ElGamal ciphertexts of  $g^{w_{j_k}}$  for each  $j_k \in J$ :  $(c_{k,1}, c_{k,2}) = (g^{r_k}, g^{w_{j_k} h^{r_k}})$ , for  $1 \leq k \leq |J|$ .

It then forms  $x'$  and  $w'$  that allow us to express the following relation  $R$  as a special case of  $R_{\text{eqrep}}$ :

$$R = \{x', w' \mid x' = (x, \{(c_{j_k,1}, c_{j_k,2})\}) \text{ and } w' = (w, w') \text{ where } w' = (r_1, \dots, r_{|J|}) \text{ such that for } 1 \leq k \leq |J|, (c_{k,1}, c_{k,2}) = (g^{r_k}, g^{w_{j_k} h^{r_k}})\}$$

In order to express  $x'$  and  $w'$  as a statement and witness for  $R_{\text{eqrep}}$ , form them as follows:  $x' = (\mathbb{G}, \{x'_i, \{g'_{i,1}, \dots, g'_{i,m'}\}\}_{i=1}^{n'})$ , where

$$\begin{aligned} n' &= n + 2|J|, m' = m + |J| \\ \text{For } 1 \leq i \leq n, x'_i &= x_i, \text{ and for } 1 \leq j \leq m, g'_{i,j} = g_{i,j}, \text{ and for } m < j \leq m + |J|, \\ &g'_{i,j} = 1. \\ \text{For } 1 \leq k \leq |J|, x'_{n+2(k-1)+1} &= c_{k,1}, g'_{n+2(k-1)+1, m+k} = g, \text{ and for } \ell \neq m+k, \\ &1 \leq \ell \leq m + |J|, g'_{n+2(k-1)+1, \ell} = 1. \\ \text{For } 1 \leq k \leq |J|, x'_{n+2k} &= c_{k,2}, g'_{n+2k, j_k} = g, g'_{n+2k, m+k} = h, \text{ and for } \ell \notin \\ &\{j_k, m+k\}, 1 \leq \ell \leq m + |J|, g'_{n+2(k-1)+1, \ell} = 1. \end{aligned}$$

Set  $w' = (w_1, \dots, w_m, r_1, \dots, r_k)$ . Using the algorithm  $\text{PS}_{\text{eqrep}}^{\text{S}}$ , compute  $\pi_{\text{eqrep}} \leftarrow \text{PoK}_{\Psi_{\text{eqrep}}} \left\{ w' : R_{\text{eqrep}}(x', w') \right\}$ , and output  $\pi = (\{(c_{k,1}, c_{k,2})\}, \pi_{\text{eqrep}})$ .

$\text{V}$  works as follows: on input the statement  $x$ , and the proof  $\pi = (\{(c_{k,1}, c_{k,2})\}, \pi_{\text{eqrep}})$ , first compute  $x'$  exactly the same way as the prover's algorithm  $\text{P}$  did. Then output  $\text{VS}_{\text{eqrep}}^{\text{S}}(x', \pi_{\text{eqrep}})$ .

**Theorem 1.** *Let the relation  $R_{\text{eqrep}} = \{(x, w)\}$  be an equality of discrete logarithm representations relation. For any  $J \subseteq [m]$ , let  $f(J, w) = \{g^{w_j} : j \in J\}$ . The proof system  $\Psi = (\text{S}, \text{P}, \text{V})$  is an  $f(J, \cdot)$ -BB-PSL simulation-extractable NIZK proof system in the random-oracle model.*

*Proof (Sketch).* We need to describe the setup simulator, the proof simulator, the extractor trapdoor and the two extractors.

- $\text{SimS}(\text{state}, m) \rightarrow (\text{state}, h', \tau_{\text{Ext}})$ : On input the description of a group  $\mathbb{G}$  with generator  $g$  of order  $q$ , sample  $\tau_{\text{Ext}} \leftarrow \mathbb{Z}_q$  and output the hash value that will be interpreted as the element  $g^{\tau_{\text{Ext}}}$  of  $\mathbb{G}$ ; we can think of this as the public key of the ElGamal cryptosystem for secret key  $\tau_{\text{Ext}}$ . On other inputs simulate the random oracle faithfully.
- $\text{Sim}(\text{state}, \mathbf{x}) \rightarrow (\text{state}, \pi)$ : On input  $\mathbf{x}$ , the simulator extends  $\mathbf{x}$  with random ElGamal ciphertexts to  $\mathbf{x}'$ , chooses  $c \leftarrow \mathbb{Z}_q$ ,  $s_j \leftarrow \mathbb{Z}_q$  for  $1 \leq j \leq m + |J|$ , and sets  $d_i = (\prod_{j=1}^{m+k} g_{i,j}^{s_j}) / x_i^c$  for  $1 \leq i \leq n + 2|J|$ . He then sets  $\text{com} = (d_1, \dots, d_{n+2|J|})$ , stores  $H[\mathbf{x}, \text{com}] = c$  in  $\text{state}$ , sets  $\text{chal} = c$ , and  $\text{res} = (s_1, \dots, s_m)$  and return  $(\text{chal}, \text{res})$ .
- $\text{Ext}^{\text{BB}(\mathcal{A})}(\mathcal{Q}_S, \mathbf{x}, \pi) \rightarrow \mathbf{w}$ : Parse  $\pi$  as  $(\text{chal}, \text{res})$  and compute  $\text{com}$  as  $\text{Sim}$ . Rewind  $\text{BB}(\mathcal{A})$  to the point where it queried the random oracle on  $(\mathbf{x}, \text{com})$  and provide it fresh random results. Repeat until it obtains two accepting transcripts for the same  $\text{com} = (d_1, \dots, d_{n+2|J|})$  and then run the extractor of the  $\Sigma$ -protocol to obtain  $\mathbf{w}'$ . Remove the last  $k$  elements to obtain  $\mathbf{w}$ .
- $\text{ExtSL}(\mathcal{Q}_S, \mathbf{x}, \pi) \rightarrow f(J, \mathbf{w})$ : Parse  $\mathbf{x}$  as  $(\mathbb{G}, \{x_i, \{g_{i,1}, \dots, g_{i,m}\}_{i=1}^{n+2|J|})$ , obtain  $\tau_{\text{Ext}}$  from the entry  $(\mathbb{G}, h, \tau_{\text{Ext}})$  of  $\mathcal{Q}_S$ . Interpret the last  $2|J|$  elements  $x_i$  as ElGamal ciphertext and decrypt them to obtain  $f(J, \mathbf{w})$ .  $\square$

### 3 Definition of Security of $f$ -Blueprint Scheme

Our scheme features three parties: an auditor, a set of users, and a set of recipients. It is tied to a non-interactive commitment scheme  $(\text{CSetup}, \text{Commit})$ ; let  $\text{cpar}$  be the parameters of the commitment scheme output by  $\text{CSetup}$ . The auditor  $A$  has private input  $x$  and publishes a commitment  $C_A = \text{Commit}_{\text{cpar}}(x)$ . The user has private data  $y$  and publishes a commitment  $C = \text{Commit}_{\text{cpar}}(y)$ . For example,  $x$  could be a list and  $y$  could be the user's attributes in a credential system. The auditor creates a key pair  $(\text{pk}_A, \text{sk}_A)$  corresponding to its input  $x$ , and the user can escrow its private data  $y$  under  $\text{pk}_A$  to obtain an escrow  $Z$ . We require that  $Z$  decrypts (with the help of  $\text{sk}_A$ ) to  $f(x, y)$  for a function  $f$  that all parties have agreed upon in advance. In the definition, we do not restrict  $f$ : it can be any efficiently computable function. Moreover, an escrow recipient  $R$  can verify that indeed  $Z$  was computed correctly for the given  $\text{pk}_A$  and  $C$ . Similarly, a privacy-conscious user can verify that indeed  $\text{pk}_A$  was computed correctly for the given warrants data commitment  $C_A$ .

**Definition 3.** *An  $f$ -blueprint scheme tied to a non-interactive commitment scheme  $(\text{CSetup}, \text{Commit})$  consists of the following probabilistic polynomial time algorithms:*

$\text{Setup}(1^\lambda, \text{cpar}) \rightarrow \Lambda$ : is the algorithm that sets up the public parameters  $\Lambda$ . It takes as input the security parameter  $1^\lambda$  and the commitment parameters  $\text{cpar}$  output by  $\text{CSetup}(1^\lambda)$ ; to reduce the number of inputs to the rest of the algorithms,  $\Lambda$  includes  $1^\lambda$  and  $\text{cpar}$ ; we will also write  $\text{Commit}$  instead of  $\text{Commit}_{\text{cpar}}$  to reduce notational overhead.

$\text{KeyGen}(\Lambda, x, r_A) \rightarrow (\text{pk}_A, \text{sk}_A)$ : is the key generation algorithm for auditor  $A$ . It takes in input  $1^\lambda$ , parameters  $\Lambda$ , and values  $(x, r_A)$ , and outputs the key pair  $(\text{pk}_A, \text{sk}_A)$ . The values  $(x, r_A)$  define a commitment  $C_A$ . This allows to integrate  $\text{KeyGen}$  into larger systems.<sup>1</sup>

$\text{VerPK}(\Lambda, \text{pk}_A, C_A) \rightarrow 1$  or  $0$ : is the algorithm that, on input the auditor’s public key  $\text{pk}_A$  and a commitment  $C_A$ , verifies that the warrant public key was computed correctly for the commitment  $C_A$ .

$\text{Escrow}(\Lambda, \text{pk}_A, y, r) \rightarrow Z$ : is the algorithm that, on input the values  $(y, r)$  outputs an escrow  $Z$  for commitment  $C = \text{Commit}(y; r)$ .

$\text{VerEscrow}(\Lambda, \text{pk}_A, C, Z) \rightarrow 1$  or  $0$ : is the algorithm that, on input the auditor’s public key  $\text{pk}_A$ , a commitment  $C$ , and an escrow  $Z$ , verifies that the escrow was computed correctly for the commitment  $C$ .

$\text{Decrypt}(\Lambda, \text{sk}_A, C, Z) \rightarrow f(x, y)$  or  $\perp$ : is the algorithm that, on input the auditor’s secret key  $\text{sk}_A$ , a commitment  $C$  and an escrow  $Z$  such that  $\text{VerEscrow}(\Lambda, \text{pk}_A, C, Z) = 1$ , decrypts the escrow. Our security properties will ensure that it will output  $f(x, y)$  if  $C$  is a commitment to  $y$ .

**Definition 4 (Secure blueprint).** *An  $f$ -blueprint scheme  $\text{Blu} = (\text{Setup}, \text{KeyGen}, \text{VerPK}, \text{Escrow}, \text{VerEscrow}, \text{Decrypt})$  tied to commitment scheme  $(\text{CSetup}, \text{Commit})$  constitutes a secure  $f$ -blueprint scheme if it satisfies the following properties:*

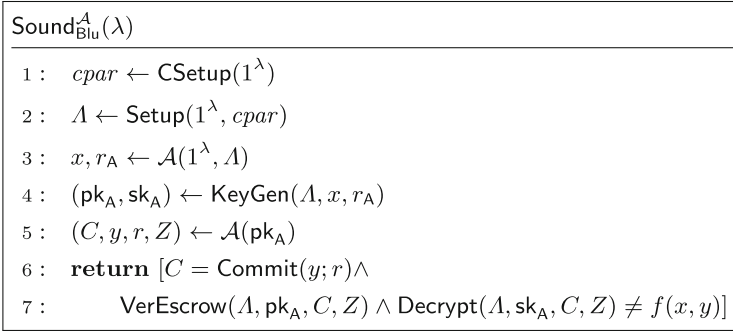
**Correctness of VerPK and VerEscrow:** Values  $(\text{cpar}, \text{pk}_A, C_A, C, Z)$  are generated honestly if: (1)  $\text{cpar}$  is generated by  $\text{CSetup}(1^\lambda)$ ; (2)  $\Lambda$  is generated by  $\text{Setup}(1^\lambda, \text{cpar})$ ; (3)  $\text{pk}_A$  is the output of  $\text{KeyGen}(\Lambda, x, r_A)$ ; (4)  $C_A = \text{Commit}_{\text{cpar}}(x; r_A)$ ; (5)  $C = \text{Commit}_{\text{cpar}}(y; r)$ ; (6)  $Z$  is generated by  $\text{Escrow}(\Lambda, \text{pk}_A, y, r)$ . For honestly generated values  $(\text{cpar}, \text{pk}_A, C_A, C, Z)$ , we require that algorithms  $\text{VerEscrow}$  and  $\text{VerPK}$  accept with probability 1.

**Correctness of Decrypt:** Similarly, we require for honestly generated  $(\text{cpar}, \text{pk}_A, \text{sk}_A, C, Z)$  that with overwhelming probability  $\text{Decrypt}(\Lambda, \text{sk}_A, C, Z) = f(x, y)$ .

**Soundness:** Let  $C_A$  and  $C$  be commitments whose openings  $(x, r_A)$  and  $(y, r)$  are known to the adversary. Let  $(\text{pk}_A, \text{sk}_A) \leftarrow \text{KeyGen}(\Lambda, x, r_A)$  be honestly derived keys. Soundness guarantees that any  $\text{pk}_A, Z$  pair that passes  $\text{VerEscrow}(\Lambda, \text{pk}_A, C, Z)$  will decrypt to  $f(x, y)$  with overwhelming probability. More formally, for all PPT adversaries  $\mathcal{A}$  involved in the experiment in Fig. 3, there exists a negligible function  $\nu$  such that:  $\text{Adv}_{\mathcal{A}, \text{Blu}}^{\text{sound}}(\lambda) = \Pr \left[ \text{Sound}_{\text{Blu}}^{\mathcal{A}}(\lambda) = 1 \right] = \nu(\lambda)$

**Blueprint Hiding:** We want to make sure that  $\text{pk}_A$  just reveals that  $x$  is a valid first argument to  $f$  (i.e. this may possibly reveal the size of  $x$  or an upper bound on its size). Otherwise,  $x$  is hidden even from an adversary who (1) may already know a lot of information about  $x$  a-priori; and (2) has oracle access to  $\text{Decrypt}(\Lambda, \text{sk}_A, \cdot, \cdot)$ .

<sup>1</sup> E.g.,  $A$  can prove that  $x$  does not contain journalists, but does contain all Russian oligarchs on the OFAC’s sanctions list. <https://home.treasury.gov/policy-issues/financial-sanctions>.



**Fig. 3.** Experiments  $\text{Sound}_{\text{Blu}}^A(\lambda)$

We formalize this security property by requiring that there exist a simulator  $\text{Sim} = (\text{SimSetup}, \text{SimKeygen}, \text{SimDecrypt})$  such that a PPT adversary cannot distinguish between the following two games: the “real” game in which  $\Lambda$  is chosen honestly, the public key  $pk_A$  is computed correctly for adversarially chosen  $x, r_A$ , and the adversary’s decryption queries  $(C, Z)$  are answered with  $\text{Decrypt}(\Lambda, sk_A, C, Z)$ ; and the “ideal” game in which  $\Lambda$  is computed using  $\text{SimSetup}$ , the public key  $pk_A$  is computed using  $\text{SimKeygen}$  independently of  $x$  (although with access to the commitment  $C_A$ ), and the adversary’s decryption query  $Z_i$  is answered by first running  $\text{SimDecrypt}$  to obtain enough information about the user’s data  $y_i$  to be able to compute  $f(x, y_i)$ . When we say “enough information,” we mean that  $\text{SimDecrypt}$  obtains  $y_i^* = g(y_i)$  for some function  $g$  such that  $f(x, y) = f^*(x, g(y))$  for an efficiently computable  $f^*$ , for all possible inputs  $(x, y)$ <sup>2</sup>.

More formally, for all probabilistic poly-time adversaries  $\mathcal{A}$  involved in the game described in Fig. 4, the advantage function satisfies:

$$\text{Adv}_{\mathcal{A}, \text{Sim}}^{\text{bh}}(\lambda) = \left| \Pr \left[ \text{BHreal}_{\text{Blu}}^A(\lambda) = 0 \right] - \Pr \left[ \text{BHideal}_{\text{Blu}, \text{Sim}}^A(\lambda) = 0 \right] \right| = \nu(\lambda)$$

for some negligible  $\nu$ .

**Privacy Against Dishonest Auditor:** There exists a simulator such that the adversary’s views in the following two games are indistinguishable:

1. **Real Game:** The adversary generates the public key and the data  $x$  corresponding to this public key, honest users follow the Escrow protocol using adversarial inputs and openings.
2. **Privacy-Preserving Game:** The adversary generates the public key and the data  $x$  corresponding to this public key. Next, for adversarially chosen inputs and openings, the users run a simulator algorithm that depends only on the commitment and  $f(x, y)$  but is independent of the commitment openings.

<sup>2</sup> For example, if  $x$  is a list  $(x_1, \dots, x_n)$  and  $f(x, y)$  checks if  $y = x_i$  for some  $i$ ,  $g(y)$  can be a one-way permutation: in order to determine whether  $y$  is on the list, it is sufficient to compute  $g(x_j)$  and compare it to  $y^* = g(y)$ .

$\text{BHreal}_{\text{Blu}}^A(\lambda)$	$\text{BHideal}_{\text{Blu,Sim}}^A(\lambda)$
1 : $cpar \leftarrow \text{CSetup}(1^\lambda)$	1 : $cpar \leftarrow \text{CSetup}(1^\lambda)$
2 : $\Lambda \leftarrow \text{Setup}(1^\lambda, cpar)$	2 : $(\Lambda, \text{state}) \leftarrow \text{SimSetup}(1^\lambda, cpar)$
3 : $(x, r_A, \text{state}_A) \leftarrow \mathcal{A}(1^\lambda, \Lambda)$	3 : $(x, r_A, \text{state}_A) \leftarrow \mathcal{A}(1^\lambda, \Lambda)$
4 :	4 : $dsim \leftarrow ( x , \text{Commit}(x; r_A))$
5 : $(pk_A, sk_A) \leftarrow \text{KeyGen}(\Lambda, x, r_A)$	5 : $(pk_A, sk_A) \leftarrow \text{SimKeygen}(1^\lambda, \text{state}, dsim)$
6 : <b>return</b> $\mathcal{A}^{\text{O}_0(pk_A, sk_A, \cdot, \cdot)}(pk_A, \text{state}_A)$	6 : <b>return</b> $\mathcal{A}^{\text{O}_1(pk_A, \text{state}, x, \cdot)}(pk_A, \text{state}_A)$
<hr/> $\text{O}_0(pk_A, sk_A, C, Z)$	<hr/> $\text{O}_1(pk_A, \text{simtrap}, x, C, Z)$
1 : <b>if</b> $\neg \text{VerEscrow}(\Lambda, pk_A, C, Z)$	1 : <b>if</b> $\neg \text{VerEscrow}(\Lambda, pk_A, C, Z)$
2 : <b>return</b> $\perp$	2 : <b>return</b> $\perp$
3 : <b>return</b> $\text{Decrypt}(\Lambda, sk_A, C, Z)$	3 : $y^* \leftarrow \text{SimDecrypt}(\text{state}, C, Z)$
	4 : <b>return</b> $f(x, y) = f^*(x, y^*)$

**Fig. 4.** Experiments  $\text{BHreal}_{\text{Blu}}^A(\lambda)$  and  $\text{BHideal}_{\text{Blu,Sim}}^A(\lambda)$

More formally, there exists algorithms  $\text{Sim} = (\text{SimSetup}, \text{SimEscrow})$  such that, for any PPT adversary  $\mathcal{A}$  involved in the game described in Fig. 5, the following equation holds for some negligible function  $\nu$ :

$$\text{Adv}_{\mathcal{A}, \text{Blu, Sim}}^{\text{pada}}(\lambda) = \left| \Pr \left[ \text{PADA}_{\text{Blu, Sim}}^{\mathcal{A}, 0}(\lambda) = 1 \right] - \Pr \left[ \text{PADA}_{\text{Blu, Sim}}^{\mathcal{A}, 1}(\lambda) = 1 \right] \right| = \nu(\lambda)$$

**Privacy with Honest Auditor:** There exists a simulator  $\text{Sim}$  such that the adversary’s views in the following two games are indistinguishable:

1. **Real Game:** The honest auditor generates the public key on input  $x$  provided by the adversary, and honest users follow the **Escrow** protocol on input adversarially chosen openings.
2. **Privacy-Preserving Game:** The honest auditor generates the public key on input  $x$  provided by the adversary. On input adversary-generated commitments and openings, the users run a simulator that is independent of  $y$  (although with access to the commitment  $C$ ) to form their escrows.

In both of these games, the adversary has oracle access to the decryption algorithm.

We formalize these two games in Fig. 6. We require that there exists a simulator  $\text{Sim} = (\text{SimSetup}, \text{SimEscrow})$  such that, for any PPT adversary  $\mathcal{A}$  involved in the game described in the figure, the following equation holds:

$$\text{Adv}_{\text{Blu, Sim}}^{\text{pwha}}(\lambda) = \left| \Pr \left[ \text{PWhA}_{\text{Blu, Sim}}^{\mathcal{A}, 0}(\lambda) = 0 \right] - \Pr \left[ \text{PWhA}_{\text{Blu, Sim}}^{\mathcal{A}, 1}(\lambda) = 0 \right] \right| = \nu(\lambda)$$



$\text{PADA}_{\text{Blu,Sim}}^{\mathcal{A},b}(\lambda)$	
1 : $cpair \leftarrow \text{CSetup}(1^\lambda)$ 2 : $\Lambda_0 \leftarrow \text{Setup}(1^\lambda, cpair); (\Lambda_1, \text{state}) \leftarrow \text{SimSetup}(1^\lambda, cpair)$ 3 : $(x, r_A, pk_A, \text{state}_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda, \Lambda_b)$ 4 : <b>if</b> $\text{VerPK}(\Lambda_b, pk_A, \text{Commit}(x; r_A)) = 0$ : <b>return</b> $\perp$ 5 : <b>return</b> $\mathcal{A}^{\text{O}_b(y,r)}(\text{state}_{\mathcal{A}})$	
$\text{O}_0(y, r)$	$\text{O}_1(y, r)$
1 : <b>return</b> $\text{Escrow}(\Lambda_0, pk_A, y, r)$	1 : <b>return</b> $\text{SimEscrow}(\text{state}, \Lambda_1, pk_A, \text{Commit}(y; r),$ 2 : $f(x, y))$

$\mathcal{A}, b$

**Fig. 5.** Game  $\text{PADA}_{\text{Blu}}^{\mathcal{A},b}(\lambda)$

for some negligible function  $\nu$ .

$\text{PWHA}_{\text{Blu,Sim}}^{\mathcal{A},b}(\lambda)$	
1 : $cpair \leftarrow \text{CSetup}(1^\lambda)$ 2 : $\Lambda_0 \leftarrow \text{Setup}(1^\lambda, cpair); \Lambda_1 \leftarrow \text{SimSetup}(1^\lambda, cpair)$ 3 : $M \leftarrow []$ 4 : $x, r_A \leftarrow \mathcal{A}(1^\lambda, \Lambda_b)$ 5 : $(pk_A, sk_A) \leftarrow \text{KeyGen}(\Lambda_b, x, r_A)$ 6 : <b>return</b> $\mathcal{A}^{\text{O}_b^{\text{Escrow}(\cdot, \cdot)}, \text{O}^{\text{Decrypt}(\Lambda_b, sk_A, \cdot, \cdot)}}(pk_A)$	
$\text{O}_0^{\text{Escrow}}(y, r)$	$\text{O}_1^{\text{Escrow}}(y, r)$
1 : <b>return</b> $\text{Escrow}(\Lambda_0, pk_A, y, r)$	1 : $C = \text{Commit}(y; r)$ 2 : $Z \leftarrow \text{SimEscrow}(\text{state}, \Lambda_1, pk_A, C)$ 3 : $M[C, Z] \leftarrow f(x, y)$ 4 : <b>return</b> $Z$
$\text{O}^{\text{Decrypt}}(\Lambda_1, sk_A, C, Z)$	
1 : <b>if</b> $M[C, Z]$ is defined <b>return</b> $M[C, Z]$ 2 : <b>return</b> $\text{Decrypt}(\Lambda_1, sk_A, C, Z)$	

**Fig. 6.** Game  $\text{PWHA}_{\text{Blu,Sim}}^{\mathcal{A},b}(\lambda)$

## 4 Homomorphic Enough Encryption

**Definition 5 (Homomorphic-enough cryptosystem (HEC) for a function family).** Let  $F = \{f \mid f : \text{domain}_{f,x} \times \text{domain}_{f,y} \mapsto \text{range}_f\}$  be a set of polynomial-time computable functions. We say that the set HEC of algorithms (HECSETUP, HECENC, HECCEVAL, HECDEC, HECDIRECT) constitute a homomorphic-enough cryptosystem (HEC) for  $F$  if they satisfy the following input-output, correctness, and security requirements:

- HECSETUP( $1^\lambda$ )  $\rightarrow$  *hecp*ar is a PPT algorithm that, on input the security parameter, outputs the parameters *hecp*ar; in case there is no HECSETUP algorithm, *hecp*ar =  $1^\lambda$ .
- HECENC(*hecp*ar,  $f$ ,  $x$ )  $\rightarrow$  ( $X$ ,  $d$ ) is a PPT algorithm that, on input the parameters *hecp*ar, a function  $f \in F$ , and a value  $x \in \text{domain}_{f,x}$ , outputs an encrypted representation  $X$  of the function  $f(x, \cdot)$ , and a decryption key  $d$ .
- HECCEVAL(*hecp*ar,  $f$ ,  $X$ ,  $y$ )  $\rightarrow$   $Z$  is a PPT algorithm that, on input the parameters *hecp*ar, a function  $f \in F$ , an encrypted representation of  $f(x, \cdot)$ , and a value  $y \in \text{domain}_{f,y}$ , outputs a ciphertext  $Z$ , an encryption of  $f(x, y)$ .
- HECDEC(*hecp*ar,  $d$ ,  $Z$ )  $\rightarrow$   $z$  is a polynomial-time algorithm that, on input the parameters *hecp*ar, the decryption key  $d$ , and a ciphertext  $Z$ , decrypts  $Z$  to obtain a value  $z$ .
- HECDIRECT(*hecp*ar,  $X$ ,  $z$ )  $\rightarrow$   $Z$  is a PPT algorithm that, on input *hecp*ar, an encrypted representation  $X$  of some function, and a value  $z$ , outputs a ciphertext  $Z$ .

**HEC Correctness.** For a given adversary  $\mathcal{A}$  and HEC, let  $\text{Adv}_{\text{HEC}, \mathcal{A}}(\lambda)$  be the probability that the experiment HECCORRECT in Fig. 7 accepts. HEC is *correct* if  $\text{Adv}_{\text{HEC}, \mathcal{A}}(\lambda)$  is negligible for all PPT algorithms  $\mathcal{A}$ .

**Security of  $x$ , Security of  $x$  and  $y$  from Third Parties, and Security of DIRECTZ.** Consider Fig. 7. For a given HEC and an adversary  $\mathcal{A}$ , and for  $b \in \{0, 1\}$ , let  $p_{\mathcal{A}, b}^{\text{SECX}}(\lambda)$  be the probability that  $\mathcal{A}$  outputs 0 in experiment  $\text{SECX}_b^{\mathcal{A}}$ , let  $p_{\mathcal{A}, b}^{\text{SECXY}}(\lambda)$  be the probability that  $\mathcal{A}$  outputs 0 in experiment  $\text{SECXY}_b^{\mathcal{A}}$ , and let  $p_{\mathcal{A}, b}^{\text{DIRECTZ}}(\lambda)$  be the probability that  $\mathcal{A}$  outputs 0 in experiment  $\text{DIRECTZ}_b^{\mathcal{A}}$ .

HEC provides *security for  $x$*  if or any PPT  $\mathcal{A}$ ,  $|p_{\mathcal{A}, 0}^{\text{SECX}}(\lambda) - p_{\mathcal{A}, 1}^{\text{SECX}}(\lambda)|$  is negligible. HEC provides *security for  $x$  and  $y$  from third parties* if or any PPT  $\mathcal{A}$ ,  $|p_{\mathcal{A}, 0}^{\text{SECXY}}(\lambda) - p_{\mathcal{A}, 1}^{\text{SECXY}}(\lambda)|$  is negligible. HEC provides *security of DIRECTZ* if or any PPT  $\mathcal{A}$ ,  $|p_{\mathcal{A}, 0}^{\text{DIRECTZ}}(\lambda) - p_{\mathcal{A}, 1}^{\text{DIRECTZ}}(\lambda)|$  is negligible.

*Remark.* Why do we need HECDIRECT? It allows us to directly form a ciphertext  $Z$  that will decrypt to a specific value  $z$ . If the function  $f$  is not one-way and it is easy, given  $z$ , to sample  $x$  and  $y$  such that  $z = f(x, y)$ , then we can derive such  $Z$  by computing  $(X, d) = \text{HECENC}(\text{hecp}ar, f, x)$  and then computing  $Z = \text{HECCEVAL}(\text{hecp}ar, f, X, y)$ . But in general, it is helpful (for some applications) to have a separate algorithm  $\text{HECDIRECT}(\text{hecp}ar, X, z)$  such that, if  $X = \text{HECENC}(\text{hecp}ar, f, x)$ , then  $Z = \text{HECDIRECT}(\text{hecp}ar, X, z)$  decrypts to  $z$  using the decryption key that corresponds to  $X$ , i.e.  $z = \text{HECDEC}(\text{hecp}ar, d, Z)$ .

HECCORRECT <sup>A</sup> (λ)	SECXY <sub>b</sub> <sup>A</sup> (λ)
1: $hecp\text{ar} \leftarrow \text{HECSETUP}(\lambda)$ 2: $(f, x, \text{state}) \leftarrow \mathcal{A}(1^\lambda, hecp\text{ar})$ 3: <b>if</b> $f \in F, x \in \text{domain}_{f,x}$ 4: $(X, d) \leftarrow \text{HECENC}(hecp\text{ar}, f, x)$ 5: $(y, r_Z) \leftarrow \mathcal{A}(\text{state}, X)$ 6: <b>if</b> $y \in \text{domain}_{f,y}$ 7: $Z \leftarrow \text{HECEVAL}(hecp\text{ar}, f, X, y; r_Z)$ 8: <b>if</b> $\text{HECDEC}(hecp\text{ar}, d, Z) \neq f(x, y)$ 9: <b>return</b> 1 10: <b>return</b> 0 11: <b>return</b> 0 12: <b>return</b> 0	1: $hecp\text{ar} \leftarrow \text{HECSETUP}(1^\lambda)$ 2: $(f, x_0, x_1, \text{state}) \leftarrow \mathcal{A}(1^\lambda, hecp\text{ar})$ 3: <b>if</b> $f \in F, x_0, x_1 \in \text{domain}_{f,x}$ 4: $X, \_ \leftarrow \text{HECENC}(hecp\text{ar}, f, x, x_0)$ 5: $(y_0, y_1, \text{state}) \leftarrow \mathcal{A}(X, \text{state})$ 6: <b>if</b> $y_0, y_1 \in \text{domain}_{f,y}$ 7: $Z \leftarrow \text{HECEVAL}(hecp\text{ar}, f, X, y_0, y_1)$ 8: <b>return</b> $\mathcal{A}(Z, \text{state})$ 9: <b>return</b> $\mathcal{A}(\perp, \text{state})$ 10: <b>return</b> $\mathcal{A}(\perp, \text{state})$
SECX <sub>b</sub> <sup>A</sup> (λ)	DIRECTZ <sub>b</sub> <sup>A</sup> (λ)
1: $hecp\text{ar} \leftarrow \text{HECSETUP}(1^\lambda)$ 2: $(f, x_0, x_1, \text{state}) \leftarrow \mathcal{A}(1^\lambda, hecp\text{ar})$ 3: <b>if</b> $f \in F, x_0, x_1 \in \text{domain}_{f,x}$ 4: $X, \_ \leftarrow \text{HECENC}(hecp\text{ar}, f, x, x_0)$ 5: <b>return</b> $\mathcal{A}(hecp\text{ar}, X, \text{state})$ 6: <b>return</b> $\mathcal{A}(\perp, \text{state})$	1: $hecp\text{ar} \leftarrow \text{HECSETUP}(1^\lambda)$ 2: $(f, x, y, r_X, \text{state}) \leftarrow \mathcal{A}(1^\lambda, hecp\text{ar})$ 3: <b>if</b> $f \in F, x \in \text{domain}_{f,x}, y \in \text{domain}_{f,y}$ 4: $X, \_ = \text{HECENC}(hecp\text{ar}, f, x; r_X)$ 5: $Z_0 \leftarrow \text{HECEVAL}(hecp\text{ar}, f, X, y)$ 6: $Z_1 \leftarrow \text{HECDIRECT}(hecp\text{ar}, X, f(x, y))$ 7: <b>return</b> $\mathcal{A}(hecp\text{ar}, Z_b, \text{state})$ 8: <b>return</b> $\mathcal{A}(\perp, \text{state})$

Fig. 7. HEC correctness and security games

## 5 A Generic $f$ -Blueprint Scheme from HEC

We construct a privacy-preserving blueprint scheme using a commitment scheme, a homomorphic-enough cryptosystem, as well as two NIZK proof systems as building blocks. The scheme consists of the following six algorithms:

**Setup** takes  $\lambda$  and a commitment setup as input and generates  $hecp\text{ar}$  and assigns the NIZK oracles  $S_1$  and  $S_2$ . Note that when instantiated using real hash functions or reference strings both RO and CRS setups can be represented as bit-strings in implementations. **KeyGen** uses the HEC scheme to compute an encrypted representation of the function  $f(x, \cdot)$  and proves that it was computed correctly. **VerPK** verifies that  $\text{pk}_A$  was computed correctly with respect to the auditor's commitment  $C_A$ . **Escrow** homomorphically evaluates  $f(x, \cdot)$  on  $y$  to obtain a ciphertext and proves that it was formed correctly. **VerEscrow** verifies the ciphertext with respect to the user's commitment  $C$ , and **Decrypt** decrypts.

Our construction in Fig. 8 uses **VerPK** as a subroutine in **Escrow** and **VerEscrow**. To be consistent with the syntax we add  $C_A$  to  $\text{pk}_A$ . Similarly, we use **VerEscrow** in **Decrypt** and add  $\text{pk}_A$  to  $\text{sk}_A$ .

**Theorem 2.** *If HEC is a secure homomorphic-enough cryptosystem, the commitment scheme is binding, and the NIZK PoKs  $\Psi_1$  and  $\Psi_2$  are zero-knowledge*

<p><b>Setup</b>(<math>\lambda, cpar</math>)</p> <hr/> $hecpair \leftarrow \text{HECSETUP}(1^\lambda)$ <b>return</b> $\Lambda = (\lambda, cpar, hecpair, S_1, S_2)$ <p><b>KeyGen</b>(<math>\Lambda, x, r_A</math>)</p> <hr/> <b>parse</b> $\Lambda = (\lambda, cpar, hecpair, S_1, S_2)$ $(X, d) \stackrel{r_X}{\leftarrow} \text{HECENC}(hecpair, f, x)$ $C_A = \text{Commit}_{cpar}(x; r_A)$ $\pi_A \leftarrow \text{PoK}_{\psi_1}^{S_1} \left\{ (x, d, r_X, r_A) : \right.$ $\quad (X, d) = \text{HECENC}(hecpair, f, x; r_X)$ $\quad \wedge C_A = \text{Commit}_{cpar}(x; r_A) \left. \right\}$ $\text{pk}_A \leftarrow (X, C_A, \pi_A); \text{sk}_A \leftarrow (\text{pk}_A, d)$ <b>return</b> $(\text{pk}_A, \text{sk}_A)$ <p><b>VerPK</b>(<math>\Lambda, \text{pk}_A, C_A</math>)</p> <hr/> <b>parse</b> $\Lambda = (\lambda, cpar, hecpair, S_1, S_2)$ <b>parse</b> $\text{pk}_A = (X, C'_A, \pi_A)$ <b>return</b> $\text{V}_1^{S_1}((X, hecpair, f, C_A, cpar), \pi_A)$ $\wedge (C'_A = C_A)$	<p><b>Escrow</b>(<math>\Lambda, \text{pk}_A, y, r</math>)</p> <hr/> <b>parse</b> $\Lambda = (\lambda, cpar, hecpair, S_1, S_2)$ <b>parse</b> $\text{pk}_A = (X, C_A, -)$ <b>if</b> $\text{VerPK}(\Lambda, \text{pk}_A, C_A) = 0$ <b>return</b> 0 $\hat{Z} \stackrel{r, \hat{Z}}{\leftarrow} \text{HECEVAL}(hecpair, f, X, y)$ $C = \text{Commit}_{cpar}(y; r)$ $\pi_U \leftarrow \text{PoK}_{\psi_2}^{S_2} \left\{ (y, r, r_{\hat{Z}}) : \right.$ $\quad \hat{Z} = \text{HECEVAL}(hecpair, f, X, y; r_{\hat{Z}})$ $\quad \wedge C = \text{Commit}_{cpar}(y; r) \left. \right\}$ <b>return</b> $(\hat{Z}, \pi_U)$ <p><b>VerEscrow</b>(<math>\Lambda, \text{pk}_A, C, Z = (\hat{Z}, \pi_U)</math>)</p> <hr/> <b>parse</b> $\Lambda = (\lambda, cpar, hecpair, S_1, S_2)$ <b>parse</b> $\text{pk}_A = (-, C_A, -)$ <b>return</b> $\text{VerPK}(\Lambda, \text{pk}_A, C_A)$ $\wedge \text{V}_2^{S_2}((\hat{Z}, hecpair, f, X, C, cpar), \pi_U)$ <p><b>Decrypt</b>(<math>\Lambda, \text{sk}_A, C, Z = (\hat{Z}, \pi_U)</math>)</p> <hr/> <b>parse</b> $\Lambda = (\lambda, cpar, hecpair, S_1, S_2)$ <b>parse</b> $\text{sk}_A = (\text{pk}_A, d)$ <b>if</b> $\text{VerEscrow}(\Lambda, \text{pk}_A, C, Z) = 0$ <b>return</b> 0 <b>return</b> $\text{HECDEC}(hecpair, d, \hat{Z})$
--	---

**Fig. 8.** Construction of generic  $f$ -blueprint scheme

and  $BB$ -PSL simulation extractable then our generic blueprint scheme is a secure  $f$ -blueprint scheme.

Note that, our formal security theorem does not require the commitment to be hiding. It only shows, using simulation, that no additional information besides the commitment is revealed. To benefit from the hiding and privacy properties of the blueprint scheme it is, however, crucial that the transaction system employing it uses a hiding commitment scheme.

We prove correctness of  $\text{VerEscrow}$  and  $\text{VerPK}$ , correctness of  $\text{Decrypt}$ , soundness, blueprint hiding, privacy against dishonest auditor, and privacy with honest auditor in separate lemmas. The statement and proof of these lemmas are in the full version [52].

## 6 HEC from the ElGamal Cryptosystem

For a binary string  $y$  (or an integer which can be interpreted as a binary string) and an integer  $k$ , let  $\text{lobits}_k(y) = y \bmod 2^k$ ; i.e.,  $\text{lobits}_k(y)$  denotes the  $k$  least significant bits of  $y$  (or, equivalently, the corresponding integer). Let  $\text{domain}_{f,y} = \{0, 1\}^{\ell_y}$ . Let us use bold font to indicate that  $x$  is a set of values; let  $W_\ell = \{x \mid x \subseteq \text{domain}_{f,y}, |x| = \ell\}$ . Let the function family  $F_\ell = \{f_k\}$ , where  $f_k : W_\ell \times \text{domain}_{f,y} \mapsto \text{domain}_{f,y}$  is defined as follows:

$$f_k(x, y) = \begin{cases} y & \text{lobits}_k(y) \in x \\ \emptyset & \text{otherwise} \end{cases}$$

In other words, the function reveals  $y$  if  $\text{lobits}_k(y) \in x$ , and nothing otherwise.

In this section, we will use the ElGamal cryptosystem in order to construct an HEC for  $f_\ell \in F_\ell$  for any  $k, \ell$  such that  $\ell$  and  $2^{\ell_y - k}$  are polynomial in  $\lambda$ . Our cryptosystem will use a group  $G$  of prime order  $q > 2^{\ell_y}$ .

### 6.1 The ElGamalHEC Construction and Its Security

The idea of our construction ElGamalHEC for a HEC for functions  $f_k \in F_\ell$ , is that HECENC outputs the ElGamal ciphertexts of the coefficients of a random polynomial  $P$  of degree  $\ell = |x|$  whose roots are elements of  $x$ . More precisely,  $P = s \prod_{i=1}^{|x|} (\chi - x_i)$ , that is  $P = \sum_{i=0}^{|x|} P_i \chi^i$ . The randomness in  $P$  comes from the choice of the leading coefficient  $s$ . HECENC outputs the ciphertexts  $C_i \leftarrow \text{Enc}(g^{P_i})$  that encrypt the coefficients  $P_i$  of  $P$ ; these ciphertext are part of  $X$ .

Using these ciphertexts  $\{C_i\}$  and the homomorphic properties of ElGamal, HECVAL computes an encryption of  $g^{rP(\text{lobits}_k(y))+y}$  for a random  $r$ . Note that if  $\text{lobits}_k(y) \in x$ , this is just an encryption of  $g^y$ ; otherwise, it is an encryption of a random element of  $G$ . Thus, HECDEC can use the ElGamal decryption algorithm to obtain some group element  $g^z$ , and then use the fact that  $\ell$  and  $2^{\ell_y - k}$  to either recover  $y$  with exhaustive search, or determine that  $f_k(x, y) = \emptyset$ .

Figure 9 describes our construction, ElGamalHEC. Here, (KGen, Enc, Dec) are the key generation, encryption, and decryption algorithms of ElGamal. Recall that  $\oplus$  is the homomorphic operator for ciphertexts.

<u>HECENC(<math>hecp\text{ar}, f_k, \mathbf{x}</math>)</u>	<u>HECEVAL(<math>hecp\text{ar}, f_k, X, y</math>)</u>
1 : $(\text{pk}_E, \text{sk}_E) \leftarrow \text{KGen}(1^\lambda)$	1 : <b>parse</b> $X = (\text{pk}_E, \mathbf{C}_0, \dots, \mathbf{C}_{ \mathbf{x} })$
2 : $s \leftarrow_{\$} \mathbb{Z}_q^*$	2 : $eval \leftarrow \bigoplus_{i=0}^{ \mathbf{x} } (\mathbf{C}_i)^{\text{lobits}_k(y)^i}$
3 : $P \leftarrow s \prod_{i=1}^{ \mathbf{x} } (\chi - x_i)$	3 : $enc \leftarrow \text{Enc}(\text{pk}_E, g^y)$
4 : <b>for</b> $i$ <b>in</b> $\{0, \dots,  \mathbf{x} \}$	4 : $r \leftarrow_{\$} \mathbb{Z}_q$
5 : $\mathbf{C}_i \leftarrow \text{Enc}(\text{pk}_E, g^{P_i})$	5 : <b>return</b> $Z = eval^r \oplus enc$
6 : <b>return</b> $(X = (\text{pk}_E, \mathbf{C}_0, \dots, \mathbf{C}_{ \mathbf{x} }),$	<u>HECDIRECT(<math>hecp\text{ar}, X, z</math>)</u>
7 : $d = (\text{sk}_E, f_k, \mathbf{x}))$	1 : <b>parse</b> $X = (\text{pk}_E, \mathbf{C}_0, \dots, \mathbf{C}_{ \mathbf{x} })$
<u>HECDEC(<math>hecp\text{ar}, d = (\text{sk}_E, f_k, \mathbf{x}), Z</math>)</u>	2 : <b>if</b> $z = \emptyset$
1 : $D \leftarrow \text{Dec}(\text{sk}_E, Z)$	3 : $\beta \leftarrow_{\$} \mathbb{Z}_q$
2 : <b>for</b> $y$ <b>in</b> $\text{domain}_{f,y} \wedge \text{lobits}_k(y) \in \mathbf{x}$	4 : <b>return</b> $\text{Enc}(\text{pk}_E, g^\beta)$
3 : <b>if</b> $g^y = D$	5 : <b>return</b> $\text{Enc}(\text{pk}_E, g^z)$
4 : <b>return</b> $y$	
5 : <b>return</b> $\emptyset$	

**Fig. 9.** Our Construction ElGamalHEC

**Theorem 3.** *Under the decisional Diffie-Hellman assumption, ElGamalHEC constitutes a homomorphic-enough encryption for  $f_k$  any  $k, \ell$  such that  $\ell$  and  $2^{\ell y - k}$  are polynomial in  $\lambda$ , for any  $f_k \in F_\ell$ .*

We prove each of the required security properties in a separate lemma in the full version [52]. As surprisingly, one of the most challenging lemmas is **HECCorrectness** due to the adversaries control over the evaluation randomness, we reproduce it here.

**Lemma 1.** *Under the decisional Diffie-Hellman assumption, ElGamalHEC satisfies the correctness property of HEC for  $f_k$ .*

*Proof.* Let  $\mathcal{A}$  be a PPT adversary playing the HEC correctness game with ElGamalHEC. Let  $\epsilon_{\mathcal{A}}(1^\lambda)$  be the probability that the challenger accepts. Below, we (1) provide modified games  $G_0$  and  $G_1$  such that the probability that the challenger in  $G_0$  accepts is also  $\epsilon_{\mathcal{A}}(1^\lambda)$ ; (2) prove that the probability  $\epsilon'_{\mathcal{A}}(1^\lambda)$  that the challenger in  $G_1$  accepts is negligible; (3) give a reduction  $\mathcal{B}_{\text{HEC}}$  that breaks the security of the ElGamal cryptosystem with advantage  $\epsilon_{\mathcal{A}}(1^\lambda) - \epsilon'_{\mathcal{A}}(1^\lambda)$ . Since the ElGamal cryptosystem is secure under the DDH assumption, it follows that, under the DDH assumption,  $\epsilon_{\mathcal{A}}(1^\lambda)$  is negligible.

- (1) First, consider the following game  $G_0$ , which is the same as the HEC correctness game with our ElGamal instantiation, except that actual polynomial evaluation instead of homomorphic evaluation.  $G_0$  first obtains

IND-CPA $_{\mathcal{B},b}(1^\lambda)$	$\mathcal{O}_b(\text{pk}_E, m_0, m_1)$
1 : $(\text{pk}_E, \text{sk}_E) \leftarrow \text{\$ KGen}(1^\lambda)$	1 : <b>return</b> $\text{Enc}(\text{pk}_E, m_b)$
2 : <b>return</b> $\mathcal{B}^{\mathcal{O}_b(\text{pk}_E, \cdot, \cdot)}(1^\lambda, \text{pk}_E)$	
$\mathcal{B}^{\mathcal{O}_b(\cdot, \cdot)}(1^\lambda, \text{pk}_E)$ from $\mathcal{A}$ attacking correctness of ElGamalHEC	
1 : $\text{hecpa} \leftarrow \text{HECSETUP}(\lambda)$	$X \leftarrow (\text{pk}_E, \mathbf{C}_0, \dots, \mathbf{C}_{ x })$
2 : $(f, \mathbf{x}, \text{state}) \leftarrow \mathcal{A}(1^\lambda, \text{hecpa})$	$(y, r_Z) \leftarrow \mathcal{A}(\text{state}, X)$
3 : <b>if</b> $f \in F, \mathbf{x} \in \text{domain}_{f,\mathbf{x}}$	<b>if</b> $y \in \text{domain}_{f,y}$
4 : $s_0 \leftarrow \text{\$ } \mathbb{Z}_q^*$	<b>parse</b> $r_Z = (r, r_{\text{Enc}})$
5 : $s_1 \leftarrow \text{\$ } \mathbb{Z}_q^*$	$y' \leftarrow P_0(\text{lobits}_k(y))r + y$
6 : $P_0 \leftarrow s_0 \prod_{i=1}^{ \mathbf{x} } (\chi - x_i)$	<b>if</b> $(\text{lobits}_k(y') \in x) \wedge (\text{lobits}_k(y) \notin x)$
	<b>return</b> 1
7 : $P_1 \leftarrow s_1 \prod_{i=1}^{ \mathbf{x} } (\chi - x_i)$	<b>return</b> 0
	<b>return</b> 0
8 : <b>for</b> $i$ <b>in</b> $\{0, \dots,  x \}$	<b>return</b> 0
9 : $\mathbf{C}_i \leftarrow \mathcal{O}_b(g^{P_{0,i}}, g^{P_{1,i}})$	

**Fig. 10.** Reduction for part (3) of the proof of Lemma 1

$(f, x, \text{state}) \leftarrow \mathcal{A}(1^\lambda, \text{hecpa})$ ; if  $f \in F, x \in \text{domain}_{f,x}$ , then it computes  $X$  as  $\text{HECENC}(\text{hecpa}, f_k, x)$ , except that it renames  $P$  into  $P_0$  and  $s$  into  $s_0$ , i.e., it computes a polynomial  $P_0(\chi) = s_0 \prod_{i=1}^{|\mathbf{x}|} (\chi - x_i)$ , where  $\{x_i\} = x$ . Next, it invokes  $\mathcal{A}$  again to receive  $(y, r_Z) \leftarrow \mathcal{A}(\text{state}, X)$ ; from it, it computes  $y' \leftarrow P_0(\text{lobits}_k(y))r_Z + y$ . If  $(\text{lobits}(y') \in x) \wedge (\text{lobits}(y) \notin x)$ , accept. That is, instead of a homomorphic evaluation of  $P_0$  using the ciphertexts  $\mathbf{C}_0, \dots, \mathbf{C}_{|x|}$ , followed by decrypting the resulting ciphertext, it performs an actual evaluation of  $P_0$ . Observe that the probability that the challenger in  $G_0$  accepts is the same as in the original correctness game due to the correctness of homomorphic polynomial evaluation.

Second, consider the game  $G_1$  that proceeds similarly to  $G_0$ : in addition to polynomial  $P_0$  it computes a polynomial  $P_1(\chi) = s_1 \prod_{i=1}^{|\mathbf{x}|} (\chi - x_i)$  with its own random value  $s_1$  that it uses within  $\text{HECENC}$  (instead of  $P_0$ ). Thus,  $X$  consists of the ciphertexts that correspond to coefficients of  $P_1$ . Running  $\text{HECEVAL}$  followed by  $\text{HECDEC}$  on input  $y$  would correspond to homomorphically evaluating  $P_1(y)$ ; instead,  $G_1$  (like  $G_0$ ) uses  $P_0$  to compute  $y' \leftarrow P_0(\text{lobits}_k(y))r_Z + y$  and accepts if  $(\text{lobits}(y') \in x) \wedge (\text{lobits}(y) \notin x)$ .

- (2) Let us prove that the probability  $\epsilon'_A(1^\lambda)$  that the challenger in  $G_1$  accepts is negligible. The challenger will accept only if  $\text{lobits}(y) \notin x$ , so let us consider this case. Then  $P_0(\text{lobits}_k(y)) \neq 0$ , and, since  $s_0$  is random,

$y' = P_0(\text{lobits}_k(y)) \neq 0$  is independent of  $\mathcal{A}$ 's view. Thus, for any  $x \in x$ ,  $\Pr[\text{lobits}(y') = x] \approx 2^{-k}$ , thus the probability that  $G_1$  accepts is  $\approx |x| 2^{-k}$ .

- (3) We construct the reduction  $\mathcal{B}_{\text{HEC}}$  to the security of the ElGamal cryptosystem. Recall that, under the DDH assumption, the ElGamal cryptosystem is CPA-secure using the formulation of Boneh and Shoup (see Sect. 2); we use this version of (multi-instance) CPA-security in our reduction (this makes the proof simpler as it avoids the hybrid argument).  $\mathcal{B}_{\text{HEC}}$  creates both polynomials  $P_j \leftarrow s_j \prod_{i=1}^{|x|} (\chi - x_i)$ ,  $j \in \{0, 1\}$ . Let  $P_{j,i}$  be their coefficients. It obtains the encryption of the coefficients of one of these polynomials via the ElGamal challenger:  $\mathbf{C}_i \leftarrow \text{O}_b(g^{P_{0,i}}, g^{P_{1,i}})$ . This is described in more detail in Fig. 10. Observe that,  $\mathcal{B}_{\text{HEC}}^{\text{Ob}(\cdot, \cdot)}(1^\lambda, \text{pk}_E)$  creates the same view for  $\mathcal{A}$  as  $G_b$ . Therefore,  $\Pr[\text{IND-CPA}_{\mathcal{B}_{\text{HEC}}, 0}(1^\lambda) = 0] = \epsilon_{\mathcal{A}}(1^\lambda)$  and  $\Pr[\text{IND-CPA}_{\mathcal{B}_{\text{HEC}}, 1}(1^\lambda) = 0] = \epsilon'_{\mathcal{A}}(1^\lambda)$ . Since ElGamal is CPA-secure under the DDH assumption,  $\epsilon_{\mathcal{A}}(1^\lambda) - \epsilon'_{\mathcal{A}}(1^\lambda) \leq \Pr[\text{IND-CPA}_{\mathcal{B}_{\text{HEC}}, 0}(1^\lambda) = 0] - \Pr[\text{IND-CPA}_{\mathcal{B}_{\text{HEC}}, 1}(1^\lambda) = 0]$  is negligible as required.  $\square$

### 6.2 From ElGamalHEC to an Efficient Secure Blueprint Scheme

In order to use our HEC construction in Fig. 9 to construct our Generic  $f$ -blueprint scheme in Fig. 8, we need a BB simulation extractable proof system for  $\Psi_1$  to prove knowledge of the witness in the following relation:

$$\left\{ \begin{array}{l} \mathbf{x} = (X, \text{hectpar}, f, C_A, \text{cpar}), \\ \mathbf{w} = (x, d, r_X, r_A) \end{array} \middle| \begin{array}{l} (X, d) = \text{HECENC}(\text{hectpar}, f, x; r_X) \wedge \\ C_A = \text{Commit}_{\text{cpar}}(x; r_A) \end{array} \right\}$$

The building blocks of this relation are statements about the message and randomness of ElGamal encryption and the opening of Pedersen commitments that can be expressed as statements about discrete logarithms representations in  $R_{\text{eqrep}}$ . By Theorem 1, we have a BB simulation-extractable NIZK proof system for  $R_{\text{eqrep}}$  and in extension  $\Psi_1$ .

For our specific construction, we assume that the auditor's commitment  $C_A$  contains commitments to coefficients of the polynomial  $P' = \prod_{i=1}^{|x|} (\chi - x_i)$ . To prove that we encrypted some polynomial  $P = sP'$  involves proving that  $P = sP'$ . We first prove that we have properly encrypted the coefficients of  $P'$ . Then, we can exponentiate these encrypted values by  $s$ , effectively multiplying the coefficients by  $s$ .

See the full version [52] for more details.

Additionally, we require that there exists a  $f'$ -BB-PSL simulation extractable proof system for  $\Psi_2$  such that there exists an efficiently computable function  $f^*$  where  $f^*(x, f'(y)) = f(x, y)$  for all  $(f, x, y) \in F \times \text{domain}_{f,x} \times \text{domain}_{f,y}$ . Recall that  $\Psi_2$  is used to prove the following relation:

$$\left\{ \begin{array}{l} \mathbf{x} = (\hat{Z}, \text{hectpar}, f, X, C, \text{cpar}), \\ \mathbf{w} = (y, r, r_{\hat{Z}}) \end{array} \middle| \begin{array}{l} \hat{Z} = \text{HECEVAL}(\text{hectpar}, f, X, y; r_{\hat{Z}}) \wedge \\ C = \text{Commit}_{\text{cpar}}(y; r) \end{array} \right\}$$



We need a range proof to prove that  $\text{lobits}_k(y)$  is used to generate Eval in  $\hat{Z}$ . This can be done using Bulletproofs [12]. The rest of the building blocks for the relation involves statements about ElGamal encryption and Pedersen commitments, we can again be expressed as eqrep relation statement.

Theorem 1 guarantees a  $f(J, \cdot)$ -BB-PSL simulation extractable NIZK system for eqrep, and in extension  $\Psi_2$ . Recall that  $f(J, \mathbb{w}) = \{g^{w_j} : j \in J\}$ . Here, if we choose  $J$  to be a singleton containing just the index corresponding to  $y$  in  $\mathbb{w}$ , we get a  $g^y$ -BB-PSL simulation extractable NIZK system. Luckily, knowing  $x$  and  $y$  is sufficient to compute  $f(x, y)$ . Here,  $f^*(x, g^y)$  can be computed similar to HECDEC in Fig. 9. We first iterate over all  $y'$  values such that  $\text{lobits}_k(y') \in x$ . If  $g^{y'} = g^y$ , we return  $y'$ . If no such value exists, we return  $\emptyset$ . Since  $|x| 2^{l_y - k}$  is polynomial in  $\lambda$ ,  $f^*$  is efficiently computable. See full version [52] for more details.

## 7 HEC for Any $f$ from Fully Homomorphic Encryption

**Definition 6. (Circuit-private (CP) fully homomorphic encryption (FHE)).** A tuple of algorithms  $(\text{FHEKeyGen}, \text{FHEEnc}, \text{FHEDec}, \text{FHEEval})$  constitute a secure fully homomorphic public-key encryption scheme [9, 10, 46, 47] if:

**Input-output specification:**  $\text{FHEKeyGen}(1^\lambda, \Lambda)$  takes as input the security parameter and possibly system parameters  $\Lambda$  and outputs a secret key  $FHESK$  and a public key  $FHEPK$ .  $\text{FHEEnc}(FHEPK, b)$  takes as input the public key and a bit  $b \in \{0, 1\}$  and outputs a ciphertext  $c$ .  $\text{FHEDec}(FHESK, c)$  takes as input a ciphertext  $c$  and outputs the decrypted bit  $b \in \{0, 1\}$ .  $\text{FHEEval}(FHEPK, \Phi, c_1, \dots, c_n)$  takes as input a public key, a Boolean circuit  $\Phi : \{0, 1\}^n \mapsto \{0, 1\}$ , and  $n$  ciphertexts and outputs a ciphertext  $c_\Phi$ ; correctness (below) ensures that  $c_\Phi$  is an encryption of  $\Phi(b_1, \dots, b_n)$  where  $c_i$  is an encryption of  $b_i$ .

**Correctness of evaluation:** For any integer  $n$  (polynomial in  $\lambda$ ) for any circuit  $\Phi$  with  $n$  inputs of size that is polynomial in  $\lambda$ , for all  $x \in \{0, 1\}^n$ , the event that  $\text{FHEDec}(FHESK, C) \neq \Phi(x)$  where  $(FHESK, FHEPK)$  are output by  $\text{FHEKeyGen}$ ,  $c_1, \dots, c_n$  are ciphertexts where  $c_i \leftarrow \text{FHEEnc}(FHEPK, x_i)$ , and  $c_\Phi = \text{FHEEval}(FHEPK, \Phi, c_1, \dots, c_n)$ , has probability 0.

**Security:** FHE must satisfy the standard definition of semantic security.

**Compactness:** What makes fully homomorphic encryption non-trivial is the property that the ciphertext  $c_\Phi$  should be of a fixed length that is independent of the size of the circuit  $\Phi$  and of  $n$ . More formally, there exists a polynomial  $s(\lambda)$  such that for all circuits  $\Phi$ , for all  $(FHESK, FHEPK)$  output by  $\text{FHEKeyGen}(\lambda)$  and for all input ciphertexts  $c_1, \dots, c_n$  generated by  $\text{FHEEnc}(FHEPK, \cdot)$ ,  $c_\Phi$  generated by  $\text{FHEEval}(FHEPK, \Phi, c_1, \dots, c_n)$  is at most  $s(\lambda)$  bits long.

An FHE scheme is, additionally, **circuit-private** [8, 35, 46, 60] for a circuit family  $\mathcal{C}$  if for any PPT algorithm  $\mathcal{A}$  that outputs  $(R, \Phi_0, \Phi_1, (x_1, r_1), \dots, (x_n, r_n))$ ,

the probability of distinguishing the homomorphic evaluation of  $\Phi_0$  on  $\{c_i = \text{FHEEnc}(FHEPK, x_i; r_i)\}_{i \in [n]}$  where  $FHEPK$  is computed as  $\text{FHEKeyGen}(1^\lambda; R)$  cannot be distinguished from the corresponding evaluation of  $\Phi_1$  on the same ciphertexts, as long as  $\Phi_0(x_1, \dots, x_n) = \Phi_1(x_1, \dots, x_n)$ .

*Bibliographic Note.* Definitions of circuit-privacy in the literature come in different flavors; we chose the formulation that makes it easiest to prove Theorem 4 below. The strongest, malicious circuit-privacy [35, 60], is strictly stronger than what we give here; therefore, constructions that achieve it automatically achieve the definition here. Constructions of circuit-private FHE from regular FHE have been given by Ostrovsky et al. [60] and by Döttling and Dujmović [35].

Similarly, we chose to formulate correctness as perfect correctness, rather than allowing a negligible probability (over the randomness for the key generation, encryption, and evaluation) of a decryption error. Our construction below also achieves HEC from schemes that are strongly correct, i.e. where the probability of a decryption error is non-zero, but with high probability, no efficient adversary can find a public key and a set of ciphertexts and a circuit that will cause a decryption error. Achieving strong correctness from the more standard notion of correctness with overwhelming probability can be done with standard techniques, see the full version [52].

**Construction of HEC for Any  $f$  from CP-FHE.** For a Boolean function  $g : \{0, 1\}^{\ell_x} \times \{0, 1\}^{\ell_y} \mapsto \{0, 1\}$ , an  $\ell_y$ -bit string  $y$  and a value  $z \in \{0, 1\}^2$ , let  $\Phi_{y,z}^g(x)$  be the Boolean circuit that outputs  $g(x, y)$  if  $z_1 = 0$ , and  $z_2$  otherwise.

Recall that our goal is to construct a secure  $f$ -HEC scheme with a direct encryption algorithm; suppose that the length of the output of  $f$  is  $\ell$ ; for  $1 \leq j \leq \ell$ , let  $f_j(x, y)$  be the Boolean function that outputs the  $j^{\text{th}}$  bit of  $f(x, y)$ . Suppose we are given an FHE scheme that is circuit-private for the families of circuits  $\{\mathcal{C}_j\}$  defined as follows:  $\mathcal{C}_j = \{\Phi_{y,z}^{f_j} : y \in \{0, 1\}^{\ell_y}, z \in \{0, 1\}^2\}$ .

$\text{HECSETUP}(1^\lambda) \rightarrow \Lambda$  : Generate the FHE parameters  $\Lambda$ , if needed.

$\text{HECENC}(1^\lambda, \Lambda, f, x) \rightarrow (X, d)$  :  
 Generate  $(FHESK, FHEPK) \leftarrow \text{FHEKeyGen}(1^\lambda, \Lambda)$ . Let  $|x| = n$ ; set  $c_i \leftarrow \text{FHEEnc}(FHEPK, x_i)$ . Output  $X = (FHEPK, c_1, \dots, c_n)$ , and decryption key  $d = FHESK$ .

$\text{HECEVAL}(hecp\text{ar}, f, X, y) \rightarrow Z$  : Parse  $X = (FHEPK, c_1, \dots, c_n)$ . For  $j = 1$  to  $\ell$ , compute  $Z_j \leftarrow \text{FHEEval}(FHEPK, \Phi_{y,00}^{f_j}, c_1, \dots, c_n)$ . Output  $Z = (Z_1, \dots, Z_\ell)$ .

$\text{HECDEC}(hecp\text{ar}, d, Z) \rightarrow z$  : Output  $(\text{FHEDec}(d, Z_1), \dots, \text{FHEDec}(d, Z_\ell))$ .

$\text{HECDIRECT}(hecp\text{ar}, X, z) \rightarrow Z$  : Parse  $X = (FHEPK, c_1, \dots, c_n)$ . For  $j = 1$  to  $\ell$ , compute  $Z_j \leftarrow \text{FHEEval}(FHEPK, \Phi_{0^\ell, 1z_j}^{f_j}, c_1, \dots, c_n)$ . Output  $Z = (Z_1, \dots, Z_\ell)$ .

**Theorem 4.** *If  $(\text{FHEKeyGen}, \text{FHEEnc}, \text{FHEDec}, \text{FHEEval})$  is a fully-homomorphic public-key encryption scheme that is circuit-private for circuit family  $\{\mathcal{C}_f^f : f \in F\}$  defined above, then our construction above constitutes a homomorphic-enough encryption for the family  $F$ .*

*Proof.* (Sketch) Correctness follows from the perfect correctness of FHE. Security of  $x$  by semantic security of FHE. Security of  $x$  and  $y$  from third parties is also by semantic security. Finally, the security of the direct encryption algorithm follows by circuit privacy.

Combining the fact that circuit-private FHE exists if and only FHE exists, and (as we saw earlier) the fact that HEC and simulation-extractable NIZK [33] give us a secure blueprint scheme, we have the following result:

**Corollary 1.** *If fully homomorphic encryption and simulation extractable NIZK exist, then for any function  $f$ , secure  $f$ -blueprint scheme is realizable.*

**Acknowledgment.** We thank Scott Griffy and Peihan Miao for helpful discussions, and the anonymous referees for constructive feedback. This research was supported by NSF awards #2154170 and #2154941, and by grants from Meta.

## References

1. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 255–270. Springer, Heidelberg (2000). <https://doi.org/10.1007/3-540-44598-6-16>
2. Baldimtsi, F., Lysyanskaya, A.: Anonymous credentials light. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 1087–1098. ACM Press (2013). <https://doi.org/10.1145/2508859.2516687>
3. Bangerter, E., Camenisch, J., Lysyanskaya, A.: A cryptographic framework for the controlled release of certified data. In: Christianson, B., Crispo, B., Malcolm, J.A., Roe, M. (eds.) Security Protocols 2004. LNCS, vol. 3957, pp. 20–42. Springer, Heidelberg (2006). [https://doi.org/10.1007/11861386\\_4](https://doi.org/10.1007/11861386_4)
4. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: P-signatures and non-interactive anonymous credentials. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 356–374. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78524-8\\_20](https://doi.org/10.1007/978-3-540-78524-8_20)
5. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (2003). [https://doi.org/10.1007/3-540-39200-9\\_38](https://doi.org/10.1007/3-540-39200-9_38)
6. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: pitfalls of the fiat-Shamir heuristic and applications to Helios. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 626–643. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-34961-4\\_38](https://doi.org/10.1007/978-3-642-34961-4_38)
7. Boneh, D., Shoup, V.: A Graduate Course in Applied Cryptography. <https://toc.cryptobook.us/>
8. Bourse, F., Del Pino, R., Minelli, M., Wee, H.: FHE circuit privacy almost for free. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 62–89. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53008-5\\_3](https://doi.org/10.1007/978-3-662-53008-5_3)
9. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012, pp. 309–325. ACM, January 2012. <https://doi.org/10.1145/2090236.2090262>

10. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd FOCS, pp. 97–106. IEEE Computer Society Press, October 2011. <https://doi.org/10.1109/FOCS.2011.12>
11. Büinz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: towards privacy in a smart contract world. In: Boneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 423–443. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-51280-4\\_23](https://doi.org/10.1007/978-3-030-51280-4_23)
12. Büinz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press, May 2018. <https://doi.org/10.1109/SP.2018.00020>
13. Camenisch, J., Damgård, I.: Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 331–345. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-44448-3\\_25](https://doi.org/10.1007/3-540-44448-3_25)
14. Camenisch, J., Hohenberger, S., Kohlweiss, M., Lysyanskaya, A., Meyerovich, M.: How to win the clonewars: efficient periodic n-times anonymous authentication. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) Proceedings of 13th ACM Conference on Computer and Communications Security, pp. 201–210. ACM (2006)
15. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact E-cash. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 302–321. Springer, Heidelberg (2005). [https://doi.org/10.1007/11426639\\_18](https://doi.org/10.1007/11426639_18)
16. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Balancing accountability and privacy using e-cash (extended abstract). In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 141–155. Springer, Heidelberg (2006). [https://doi.org/10.1007/11832072\\_10](https://doi.org/10.1007/11832072_10)
17. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-44987-6\\_7](https://doi.org/10.1007/3-540-44987-6_7)
18. 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](https://doi.org/10.1007/3-540-36413-7_20)
19. 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](https://doi.org/10.1007/978-3-540-28628-8_4)
20. Camenisch, J., Lysyanskaya, A., Neven, G.: Practical yet universally composable two-server password-authenticated secret sharing. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012, pp. 525–536. ACM Press, October 2012. <https://doi.org/10.1145/2382196.2382252>
21. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003). [https://doi.org/10.1007/978-3-540-45146-4\\_8](https://doi.org/10.1007/978-3-540-45146-4_8)
22. 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>
23. Camenisch, J.L.: Group signature schemes and payment systems based on the discrete logarithm problem. Ph.D. thesis, ETH Zürich (1998)
24. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 34–63. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-56877-1\\_2](https://doi.org/10.1007/978-3-030-56877-1_2)

25. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) *Advances in Cryptology*, pp. 199–203. Springer, Boston (1983). [https://doi.org/10.1007/978-1-4757-0602-4\\_18](https://doi.org/10.1007/978-1-4757-0602-4_18)
26. Chaum, D.: Blind signature systems. In: Chaum, D. (ed.) *Advances in Cryptology*, pp. 153–156. Springer, Boston (1983). [https://doi.org/10.1007/978-1-4684-4730-9\\_14](https://doi.org/10.1007/978-1-4684-4730-9_14)
27. Chaum, D.: Security without identification: transaction systems to make big brother obsolete. *Commun. ACM* **28**(10), 1030–1044 (1985)
28. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Goldwasser, S. (ed.) *CRYPTO 1988*. LNCS, vol. 403, pp. 319–327. Springer, New York (1990). [https://doi.org/10.1007/0-387-34799-2\\_25](https://doi.org/10.1007/0-387-34799-2_25)
29. Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) *EUROCRYPT 1991*. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991). [https://doi.org/10.1007/3-540-46416-6\\_22](https://doi.org/10.1007/3-540-46416-6_22)
30. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) *CRYPTO 1994*. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994). [https://doi.org/10.1007/3-540-48658-5\\_19](https://doi.org/10.1007/3-540-48658-5_19)
31. Damgård, I.: On  $\sigma$ -protocols (2002). <https://www.daimi.au.dk/~ivan/Sigma.ps>
32. Damgård, I., Ganesh, C., Khoshakhlagh, H., Orlandi, C., Siniscalchi, L.: Balancing privacy and accountability in blockchain identity management. In: Paterson, K.G. (ed.) *CT-RSA 2021*. LNCS, vol. 12704, pp. 552–576. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-75539-3\\_23](https://doi.org/10.1007/978-3-030-75539-3_23)
33. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) *CRYPTO 2001*. LNCS, vol. 2139, pp. 566–598. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-44647-8\\_33](https://doi.org/10.1007/3-540-44647-8_33)
34. Diaz, J., Lehmann, A.: Group signatures with user-controlled and sequential linkability. In: Garay, J.A. (ed.) *PKC 2021, Part I*. LNCS, vol. 12710, pp. 360–388. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-75245-3\\_14](https://doi.org/10.1007/978-3-030-75245-3_14)
35. Döttling, N., Dujmovic, J.: Maliciously circuit-private FHE from information-theoretic principles. *Cryptology ePrint Archive, Report 2022/495* (2022). <https://eprint.iacr.org/2022/495>
36. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) *CRYPTO 1984*. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985). [https://doi.org/10.1007/3-540-39568-7\\_2](https://doi.org/10.1007/3-540-39568-7_2)
37. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the fiat-Shamir transform. In: Galbraith, S., Nandi, M. (eds.) *INDOCRYPT 2012*. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-34931-7\\_5](https://doi.org/10.1007/978-3-642-34931-7_5)
38. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) *CRYPTO 1986*. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). [https://doi.org/10.1007/3-540-47721-7\\_12](https://doi.org/10.1007/3-540-47721-7_12)
39. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) *CRYPTO 2005*. LNCS, vol. 3621, pp. 152–168. Springer, Heidelberg (2005). [https://doi.org/10.1007/11535218\\_10](https://doi.org/10.1007/11535218_10)
40. Frankle, J., Park, S., Shaar, D., Goldwasser, S., Weitzner, D.J.: Practical accountability of secret processes. In: Enck, W., Felt, A.P. (eds.) *USENIX Security 2018*, pp. 657–674. USENIX Association, August 2018

41. Fraser, A., Garms, L., Lehmann, A.: Selectively linkable group signatures—stronger security and preserved verifiability. In: Conti, M., Stevens, M., Krenn, S. (eds.) CANS 2021. LNCS, vol. 13099, pp. 200–221. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-92548-2\\_11](https://doi.org/10.1007/978-3-030-92548-2_11)
42. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-24676-3\\_1](https://doi.org/10.1007/978-3-540-24676-3_1)
43. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0052225>
44. Fujisaki, E., Okamoto, T.: Witness hiding protocols to confirm modular polynomial relations. In: The 1997 Symposium on Cryptography and Information Security. The Institute of Electronics, Information and Communication Engineers, Fukuoka, Japan, January 1997, sCSI97-33D
45. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-Shamir bulletproofs are non-malleable (in the algebraic group model). In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 397–426. Springer, Heidelberg (2022). [https://doi.org/10.1007/978-3-031-07085-3\\_14](https://doi.org/10.1007/978-3-031-07085-3_14)
46. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of STOC 2009, pp. 169–178 (2009)
47. 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, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40041-4\\_5](https://doi.org/10.1007/978-3-642-40041-4_5)
48. Goldwasser, S., Park, S.: Public accountability vs. secret laws: can they coexist? Cryptology ePrint Archive, Report 2018/664 (2018). <https://eprint.iacr.org/2018/664>
49. Green, M., Kaptchuk, G., Van Laer, G.: Abuse resistant law enforcement access systems. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021, Part III. LNCS, vol. 12698, pp. 553–583. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77883-5\\_19](https://doi.org/10.1007/978-3-030-77883-5_19)
50. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 406–425. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-20465-4\\_23](https://doi.org/10.1007/978-3-642-20465-4_23)
51. Kilian, J., Petrank, E.: Identity escrow. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 169–185. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0055727>
52. Kohlweiss, M., Lysyanskaya, A., Nguyen, A.: Privacy-preserving blueprints. Cryptology ePrint Archive, Paper 2022/1536 (2022). <https://eprint.iacr.org/2022/1536>
53. Libert, B., Nguyen, K., Peters, T., Yung, M.: Bifurcated signatures: folding the accountability vs. anonymity dilemma into a single private signing scheme. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021, Part III. LNCS, vol. 12698, pp. 521–552. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77883-5\\_18](https://doi.org/10.1007/978-3-030-77883-5_18)
54. Lysyanskaya, A.: Signature schemes and applications to cryptographic protocol design. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 2002

55. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym systems. In: Heys, H., Adams, C. (eds.) SAC 1999. LNCS, vol. 1758, pp. 184–199. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-46513-8\\_14](https://doi.org/10.1007/3-540-46513-8_14)
56. Lysyanskaya, A., Rosenbloom, L.N.: Universally composable sigma-protocols in the global random-oracle model. Cryptology ePrint Archive, Report 2022/290 (2022). <https://eprint.iacr.org/2022/290>
57. Maurer, U.: Unifying zero-knowledge proofs of knowledge. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 272–286. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-02384-2\\_17](https://doi.org/10.1007/978-3-642-02384-2_17)
58. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: Proceedings of 8th ACM Conference on Computer and Communications Security, pp. 116–125. ACM Press, November 2001
59. Nguyen, K., Guo, F., Susilo, W., Yang, G.: Multimodal private signatures. In: CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 792–822. Springer, Heidelberg (2022). [https://doi.org/10.1007/978-3-031-15979-4\\_27](https://doi.org/10.1007/978-3-031-15979-4_27)
60. Ostrovsky, R., Paskin-Cherniavsky, A., Paskin-Cherniavsky, B.: Maliciously circuit-private FHE. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 536–553. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-662-44371-2\\_30](https://doi.org/10.1007/978-3-662-44371-2_30)
61. Ostrovsky, R., Skeith, W.E.: Private searching on streaming data. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 223–240. Springer, Heidelberg (2005). [https://doi.org/10.1007/11535218\\_14](https://doi.org/10.1007/11535218_14)
62. 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](https://doi.org/10.1007/3-540-46766-1_9)
63. Sakai, Y., Emura, K., Hanaoka, G., Kawai, Y., Matsuda, T., Omote, K.: Group signatures with message-dependent opening. In: Abdalla, M., Lange, T. (eds.) Pairing 2012. LNCS, vol. 7708, pp. 270–294. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-36334-4\\_18](https://doi.org/10.1007/978-3-642-36334-4_18)
64. Scafuro, A.: Break-glass encryption. In: Lin, D., Sako, K. (eds.) PKC 2019, Part II. LNCS, vol. 11443, pp. 34–62. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17259-6\\_2](https://doi.org/10.1007/978-3-030-17259-6_2)
65. 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](https://doi.org/10.1007/3-540-69053-0_18)
66. Tsiounis, Y., Yung, M.: On the security of ElGamal based encryption. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 117–134. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0054019>

# Author Index

## A

Abraham, Ittai 251  
Abusalah, Hamza 282  
Applebaum, Benny 190  
Asharov, Gilad 251

## B

Ball, Marshall 3  
Basso, Andrea 405  
Boneh, Dan 499  
Boyle, Elette 159  
Bünz, Benedikt 499

## C

Chen, Binyi 499  
Chen, Megan 379  
Chiesa, Alessandro 379  
Cini, Valerio 282  
Codogni, Giulio 405  
Connolly, Deirdre 405  
Couteau, Geoffroy 159  
Cui, Hongrui 35

## D

Damgård, Ivan 129  
Dao, Quang 531  
De Feo, Luca 405

## E

Escudero, Daniel 220

## F

Fernando, Rex 98  
Fouotsa, Tako Boris 405

## G

Ganesh, Chaya 315  
Goel, Aarushi 347

Goyal, Vipul 220

Grubbs, Paul 531

Gur, Tom 379

## H

Hall-Andersen, Mathias 347, 438

## I

Ishai, Yuval 68

## J

Jain, Aayush 98

## K

Kalai, Yael Tauman 470

Kaptchuk, Gabriel 347

Khurana, Dakshita 68

Kohlweiss, Markulf 594

Komargodski, Ilan 98

Kondi, Yashvanth 315

Konstantini, Niv 190

## L

Li, Hanjun 3

Lido, Guido Maria 405

Lin, Huijia 3

Liu, Tianren 3

Lombardi, Alex 470

Lysyanskaya, Anna 594

## M

Meyer, Pierre 159

Morrison, Travis 405

## N

Narayanan, Varun 563

Nguyen, An 594

Nielsen, Jesper Buus 438



**O**

O'Connor, Jack 379  
Orlandi, Claudio 315

**P**

Pancholi, Mahak 315  
Panny, Lorenz 405  
Patil, Shravani 251  
Patra, Arpita 251  
Patranabis, Sikhar 405  
Polychroniadou, Antigoni 220  
Prabhakaran, Vinod M. 563

**R**

Ravi, Divya 129

**S**

Sahai, Amit 68  
Sangwan, Neha 563  
Siniscalchi, Luisa 129  
Song, Yifan 220

Spooner, Nicholas 347, 379  
Srinivasan, Akshayaram 68

**T**

Takahashi, Akira 315  
Tschudi, Daniel 315

**V**

Vaikuntanathan, Vinod 470

**W**

Wang, Xiao 35  
Watanabe, Shun 563  
Weng, Chenkai 220  
Wesolowski, Benjamin 405

**Y**

Yakubov, Sophia 129  
Yang, Kang 35  
Yu, Yu 35

**Z**

Zhang, Zhenfei 499